RasPi

DESIGN
BUILD
CODE

3

Get hands-on with your Raspberry Pi

16 Problems solved
Fix Your Pi

14 PRACTICAL GUIDES

Plus
Upgrade your robot with sensors

Welcome

There's one sure thing in life – if something can go wrong, it will go wrong at some point. With this in mind we've put together a Fix Your Pi feature for this third issue of **RasPi** magazine, where we tackle the most common hardware and software problems you will encounter with your Raspberry Pi. We also take the robot tutorial from issue two to a whole new level – adding in microswitches and sensors to give your creation a set of senses that will bring it to life. With 87 pages in this issue we have got lots more Raspberry Pi inspiration for you too – check out the Bioscope film player, tweet wildlife pictures and make the most of your memory with Python. Enjoy your issue and have fun!

Gavin Thomas

Deputy Editor

Get inspired
Discover the RasPi community's best projects

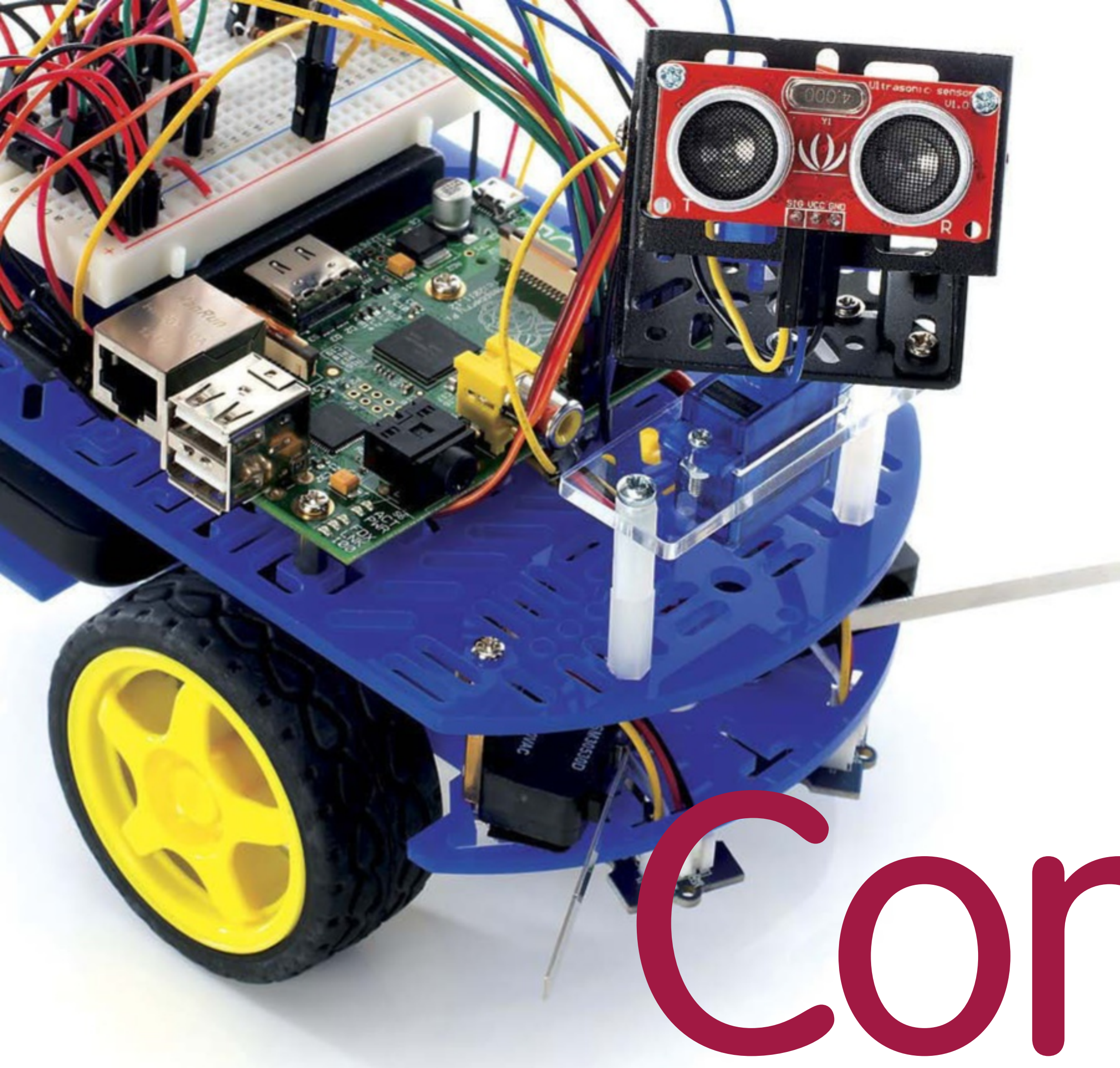
Expert advice
Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides
Learn to make and code gadgets with Raspberry Pi

From the makers of **LinuxUser & Developer**

Join the conversation at…

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk



Contents

Fix Your Pi

Troubleshooting – Hardware

A list of common Raspberry Pi hardware problems and what to do when they occur

We've all been there. We've come back to something, and for whatever reason, the thing we're trying to do just won't work! We've plugged everything back in where we thought it came from, but there's nothing. So, what do we do when it comes to this? Usually there are a few different solutions to any one problem – but sometimes there's only one way, and finding that one key to victory might not be all that easy.

> "If the picture from your Pi is noisy (e.g. screen is shaking) it's probably a dirty connection"

Maybe you've come back to your Pi after several months of it being sat in a drawer. You plug it in and boot it up and… X, Y or Z has happened!

What can we try when it just seems that everything has completely fallen over and nothing will work?

Well, hopefully your problem will be listed as one here. These are common problems for any Raspberry Pi user to come across at one point or another while using their device. Have a look through and if you're experiencing one of these issues, we've got a solution on hand for you.

Some of these things you'll want to slap yourself in the face for not thinking of – but that's all part of the joy of computers, right? The simplest of things can put you to the test for the longest of times, just ready for that eureka moment when it all finally comes together.

> "These are common problems for any Raspberry Pi user to come across at one point or another while using their device"

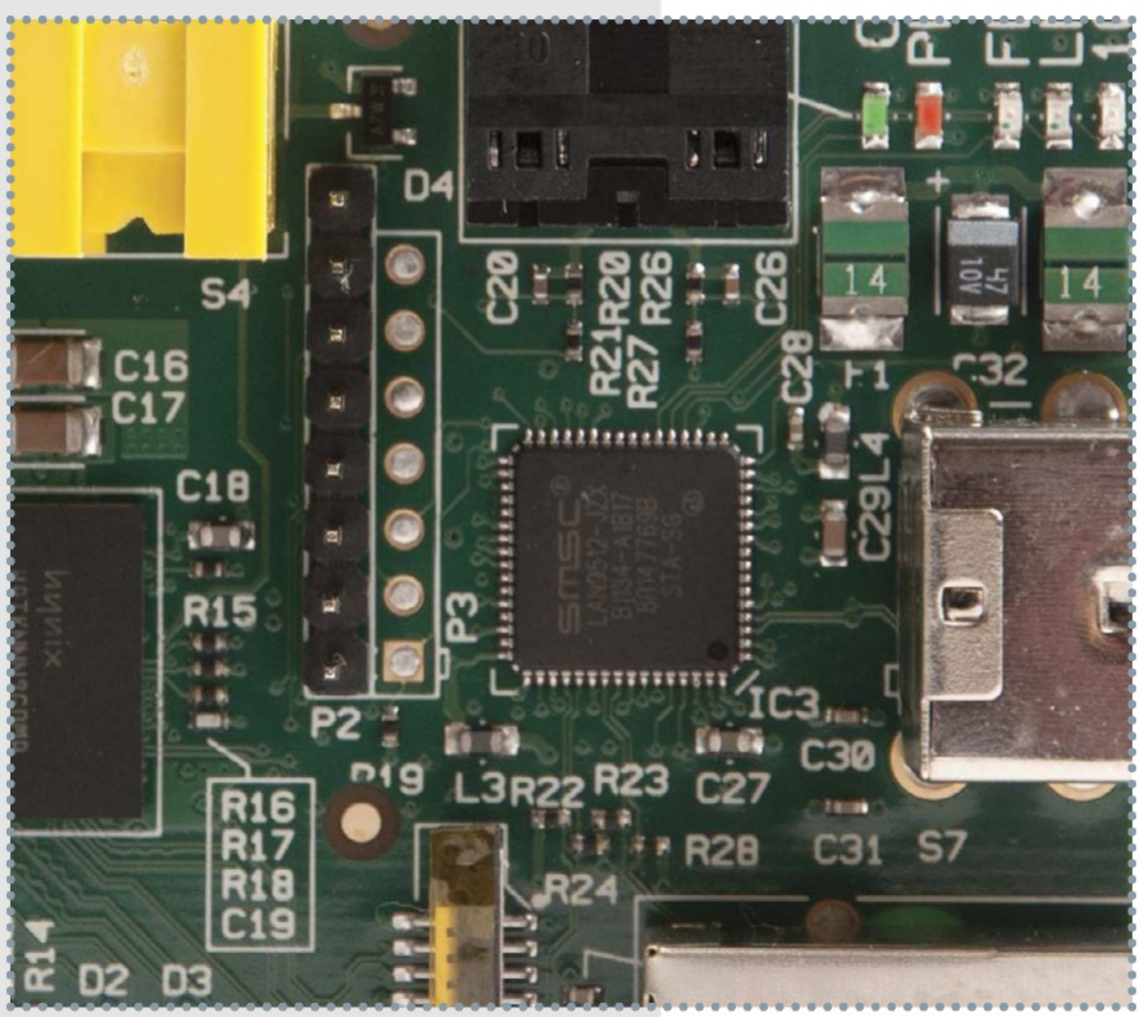
Problem 1: The green LED flashes and nothing is on screen

Solution: This is more common than you might think. It's likely down to either a blank SD card, or a lack of correct data on the SD card causing a boot failure. First, check your SD card is fully inserted. With the latest Pi firmware, if you have:

3 flashes – start.elf is missing

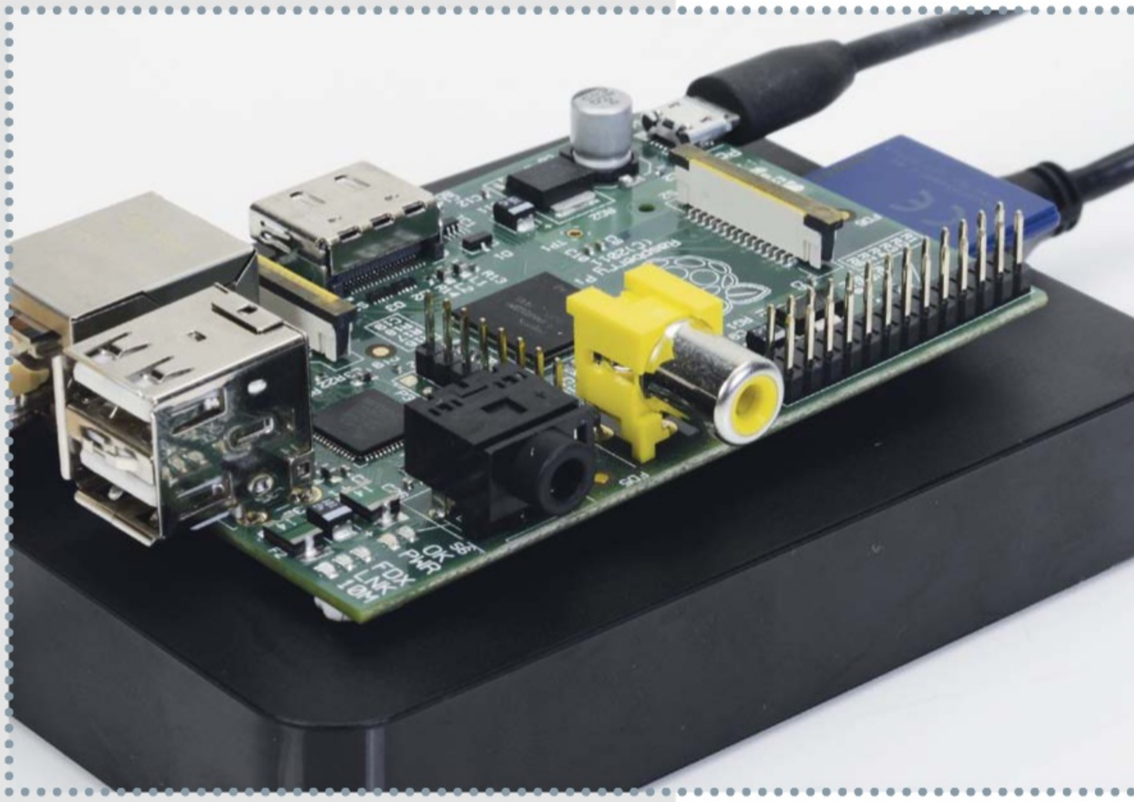
4 flashes – start.elf not launched

7 flashes – kernel.img not found



Below We'll soon have your Pi fixed and you on your way

Problem 2: I can't see my USB hard drive



Solution: This is also another really common one and is generally caused by using a non-powered portable USB hard drive in a non-powered USB hub. The easiest way to fix this is to power one side or the other. If using a portable, USB-powered hard drive, you should really use a powered USB hub.

If you definitely have power, the hard drive could be in an unreadable format. Try using FAT32.

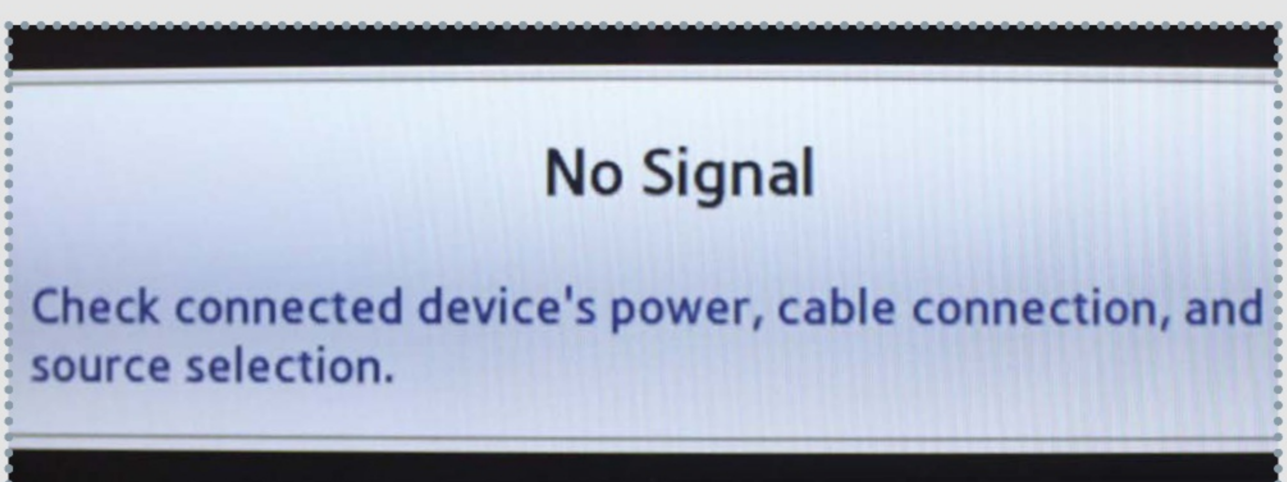
Above Is your hard drive in an unreadable format?

Problem 3: Pi shuts down or restarts a random intervals

Solution: If your Pi is particularly unstable, the first thing to check is the power supply you're using. First, swap out the cable. If you're still having problems, swap out the power supply. If your Pi is overclocked, it's also worth using the raspi-config tool to set it back to defaults. You may just be overheating it!

If setting it to default clock speeds helps, increase again gradually until you find something stable.

> "If your Pi is overclocked, it's also worth using the raspi-config tool to set it back to defaults"



Problem 4: The HDMI image from the Pi has noise

Solution: If the picture from your Pi is noisy (.e.g screen is shaking, dots or green lines moving around) then it's probably a dirty connection. Make sure the HDMI or composite cable is connected properly and free from dust and other contaminants. If you still have a problem, try another HDMI cable.

You can also try adding the following to your SD card's config.txt:

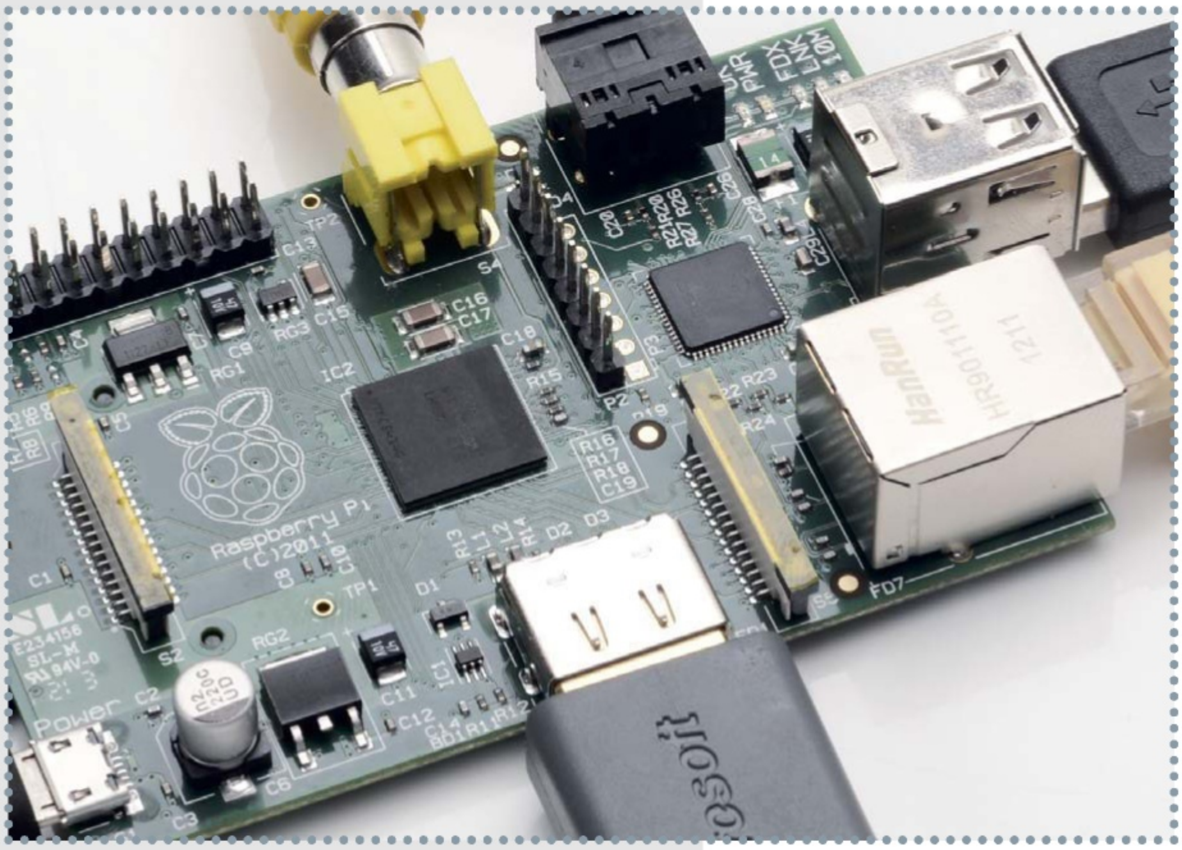
```
config_hdmi_boost=4
```

"Make sure the HDMI or composite cable is connected properly and free from dust"

Problem 5: I broke a part off!

Solution: If the part that broke off happens to be a silver cylindrical piece near the Pi's power input, this is easy enough to fix – this has been broken off by many other people due to its large surface/small mount area. This piece of the Pi reduces noise and stops power spikes. You can either reattach it (depending on your soldering skills), but you can use the Pi just fine without it – with most power supplies.

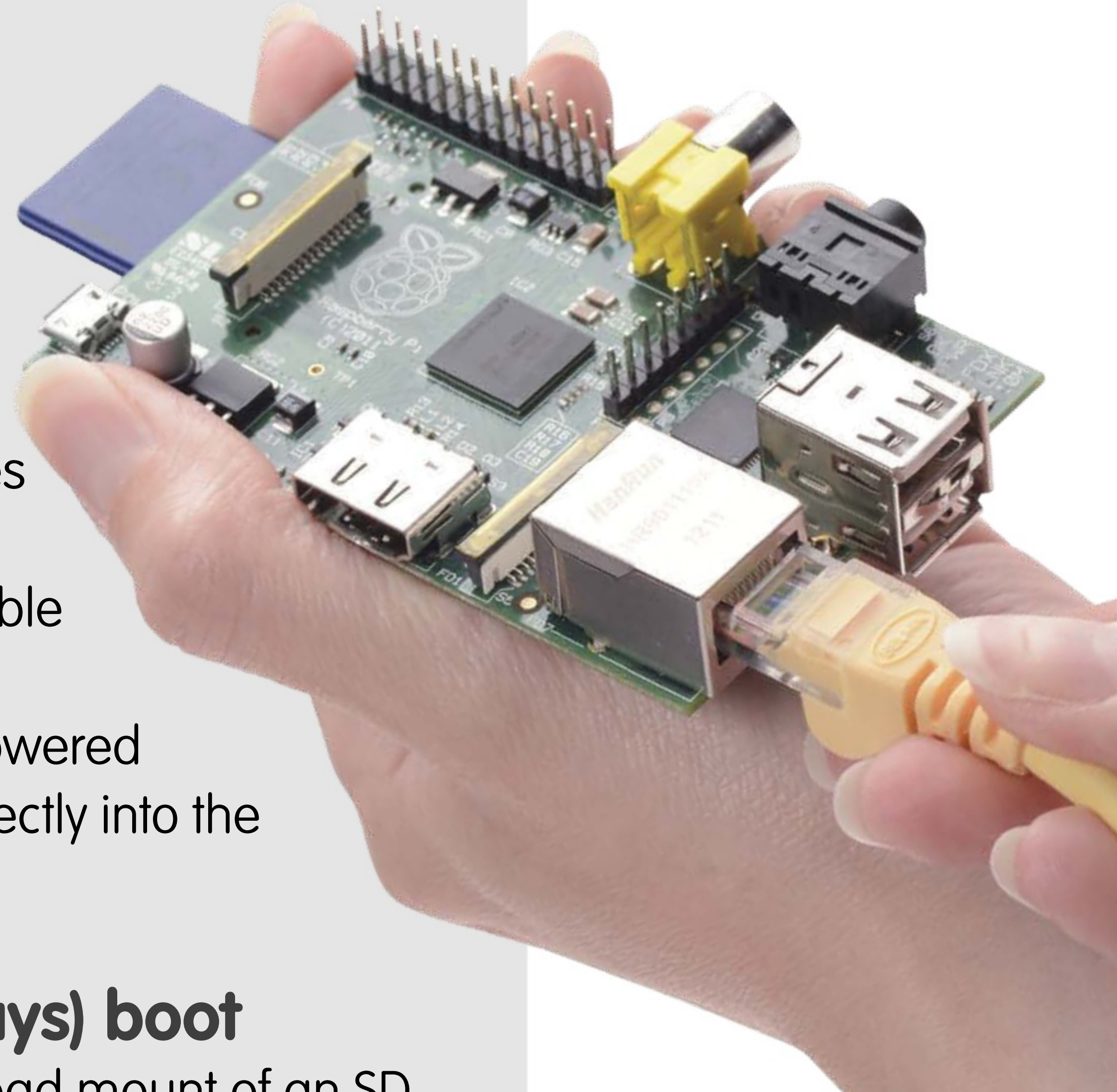
Below Don't panic if a piece of your Pi breaks off – you can fix it



Problem 6: Ethernet stops working with some USB devices

Solution: The first thing that you should do in this situation is check your power supply is working! It has been known for some power supplies to provide an inadequate source of power. First, try changing out your cable (it may just be a poor-quality cable).

Also, try the failing device in a powered USB hub – rather than plugging it directly into the Raspberry Pi.

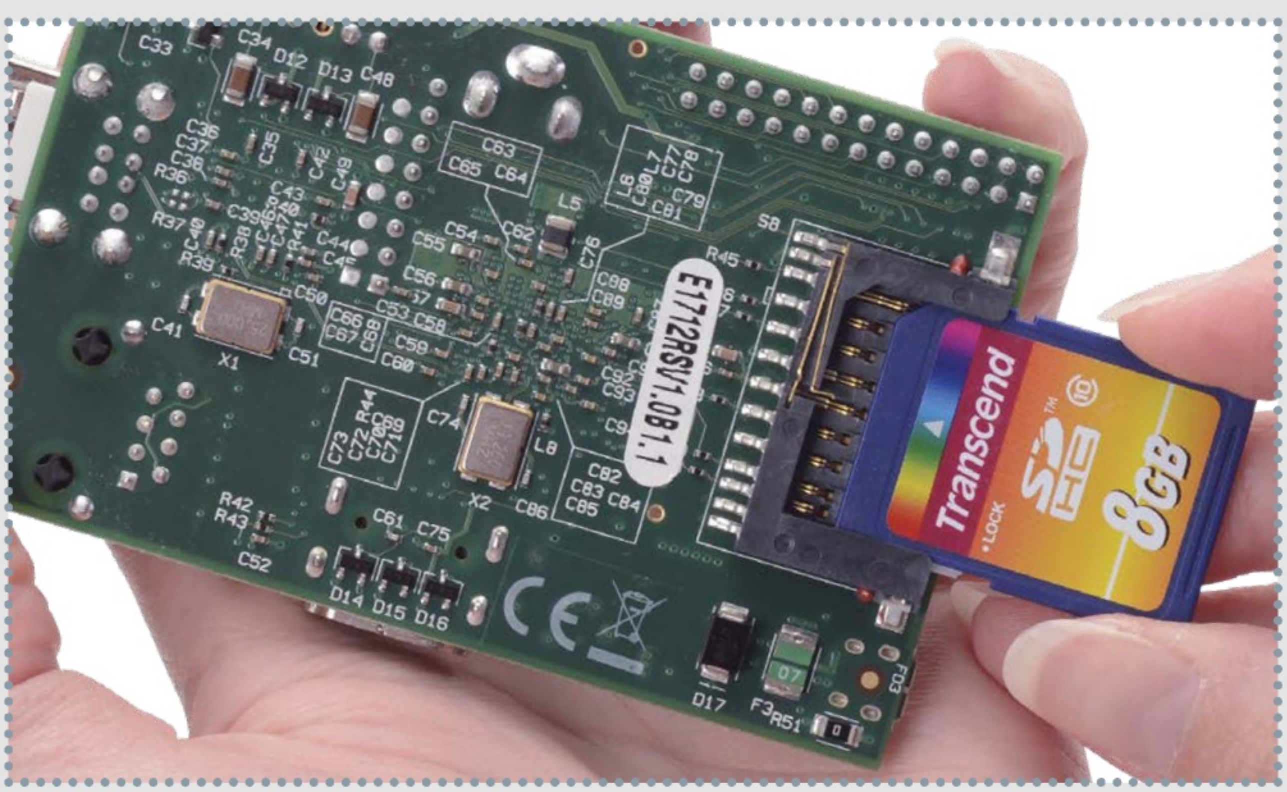


Problem 7: Pi doesn't (always) boot

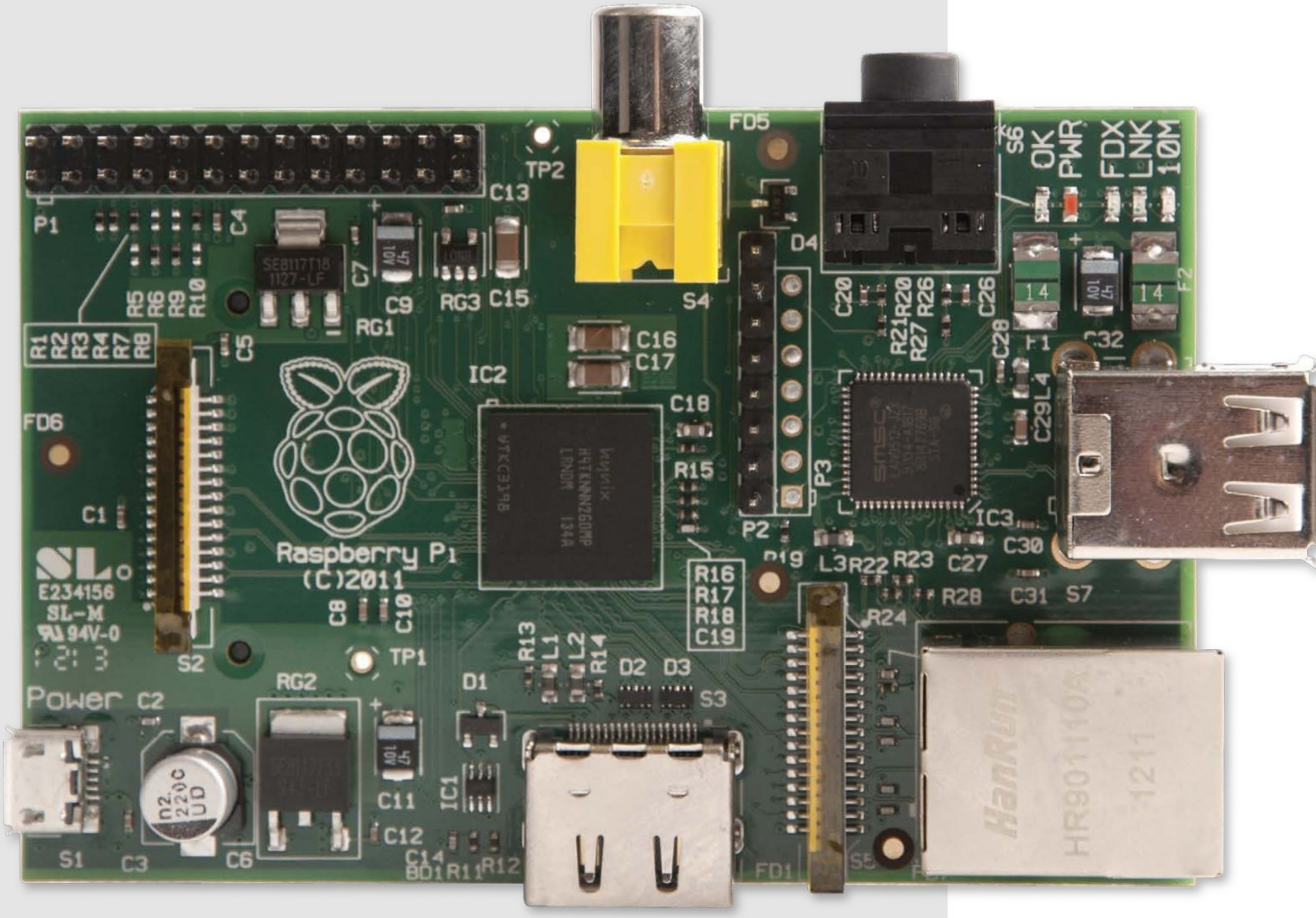
Solution: This is likely to either be a bad mount of an SD card or a power supply issue. Check your SD card: it may well be that it's not seated correctly. Take it out and put it back in straight – and make sure it goes in all the way it possibly can.

If it was definitely in correctly, try cleaning the contacts and restarting to see if it works. If this fails, try another power supply to see if it works there.

> "Try a failing device in a powered USB hub – rather than plugging it directly into the Raspberry Pi"



Left Cleaning the contacts is always a good starting point



Problem 8: Red power LED is on, nothing on display

Solution: First, check the power cable is properly seated. Check that your SD card has a valid image on it. Can you still read the SD card in the PC that the image was created on?

If you can, try booting your Raspberry Pi with just the power supply connected. Watch whether the 'OK' light flashes. If it does, add the cables back in one by one.

> "Can you still read the SD card in the PC that the image was created on?"

Above The red power LED is at the top-right of this picture, fourth from the right edge of the board

Troubleshooting – Software

A list of common Raspberry Pi software problems and what to do when they occur

Software can be just as much of a pain as hardware. The problem with software, however, is that different configuration options can lead to a really bad, frustrating time. Hardware we have attached can affect the configuration of our system and lead to further issues – sometimes the only way to go is from the ground up. You have got to think logically about these things.

Some problems will seem to have little to no correlation with their answers, but those scratch-your-head moments soon go when you realise that actually the thing is working and doing exactly what you wanted it to again. There's a lot you can do from the command line to help with any of this stuff, but things aren't always as clear as they seem.

Thankfully there are a lot of helpful tools to start making a dent in the most common problems. In these pages we feature a list of the most common software problems with their solutions.

If the worst comes to the worst then you can always rebuild it. Due to the Pi's nature, doing this is a simple reflash of a single image, rather than painstaking hours of driver and software reinstalls like some other operating systems out there.

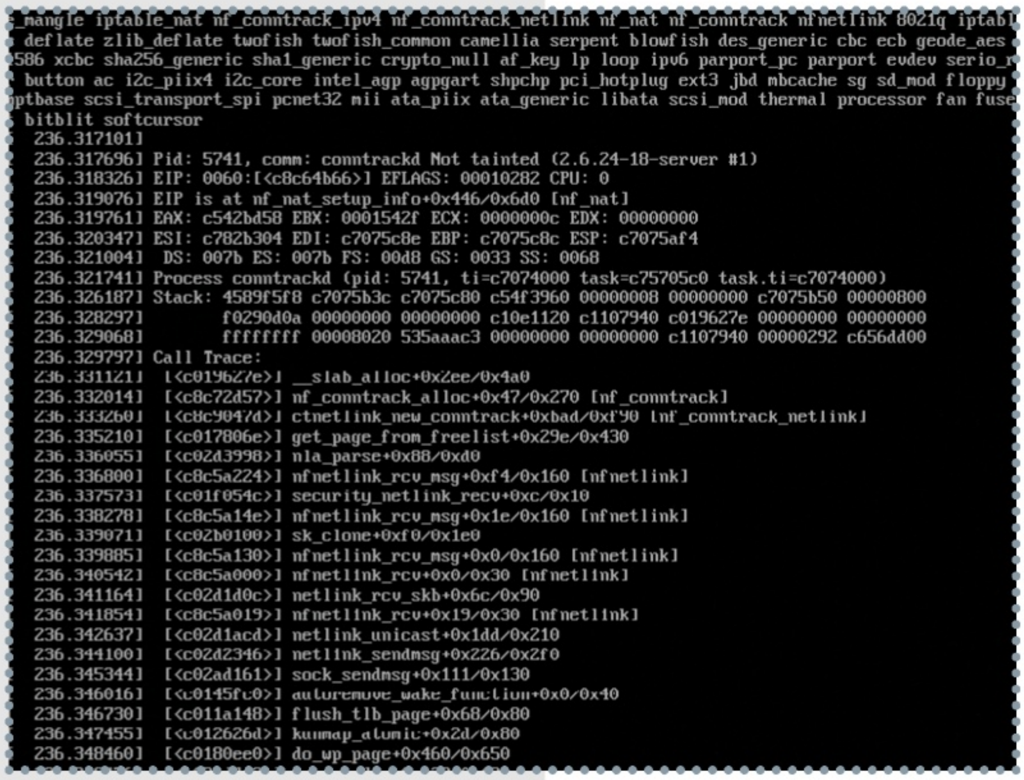
As always, be sure to keep frequent backups of your most precious data, though.

> "Sometimes the only way to go is from the ground up. You have got to think logically about these things"

Problem 9: I'm getting a kernel panic on every boot

Solution: If you're seeing kernel panic messages each time you start your Raspberry Pi, the first thing to try is booting it without any USB devices in – and adding them one by one afterwards.

If you're still getting the messages, it's likely that your flash of the SD card was unsuccessful. Reflash the card and try again. Make sure you use admin privileges to do it!



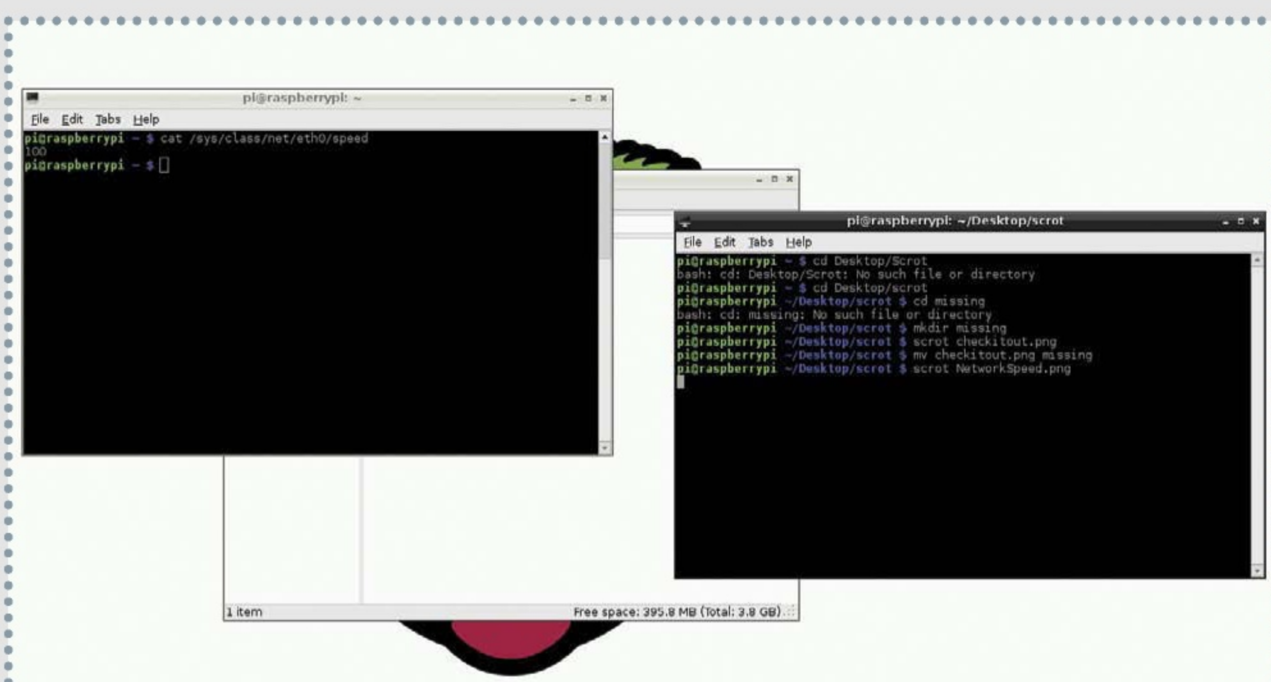
> "The first thing to try is booting it without any USB devices in – and adding them one by one afterwards"

Problem 10: My Pi's Ethernet connection is working at 10Mbit, not 100Mbit

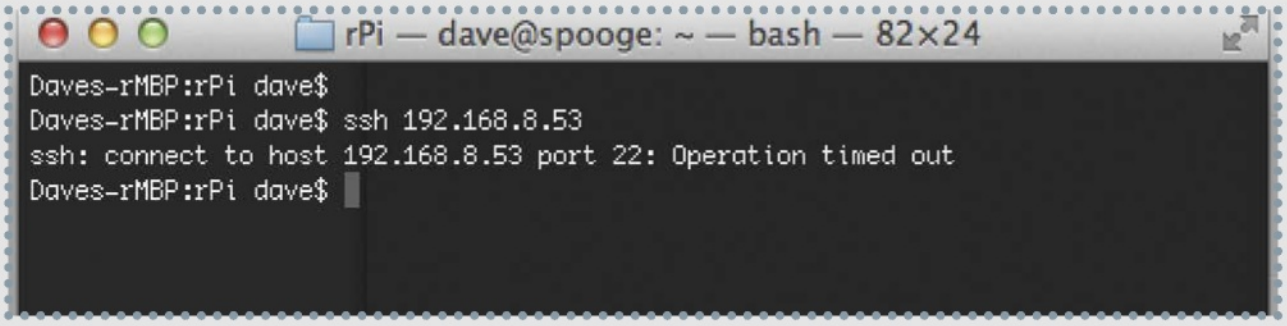
Solution: This may or may not be inaccurate. On the earlier Pi revision boards, the 100Mbit LED was mislabelled – it was actually a 10Mbit socket. On newer revision boards, however, it was correctly labelled on the newer boards that actually have a 100Mbit socket.

You can check your link speed with

```
cat /sys/class/net/eth0/speed
```



Left You can easily check your link speed via the terminal



Problem 11: I cannot SSH to my Raspberry Pi

Solution: If you're receiving a connection time-out error when you try to SSH to your Pi from another machine, it probably means that SSH access to it is disabled. You need to open up the raspi-config tool, then use the option 'ssh' to enable SSH. You should now be able to connect to your Pi. The default user/password combination is pi/raspberry.

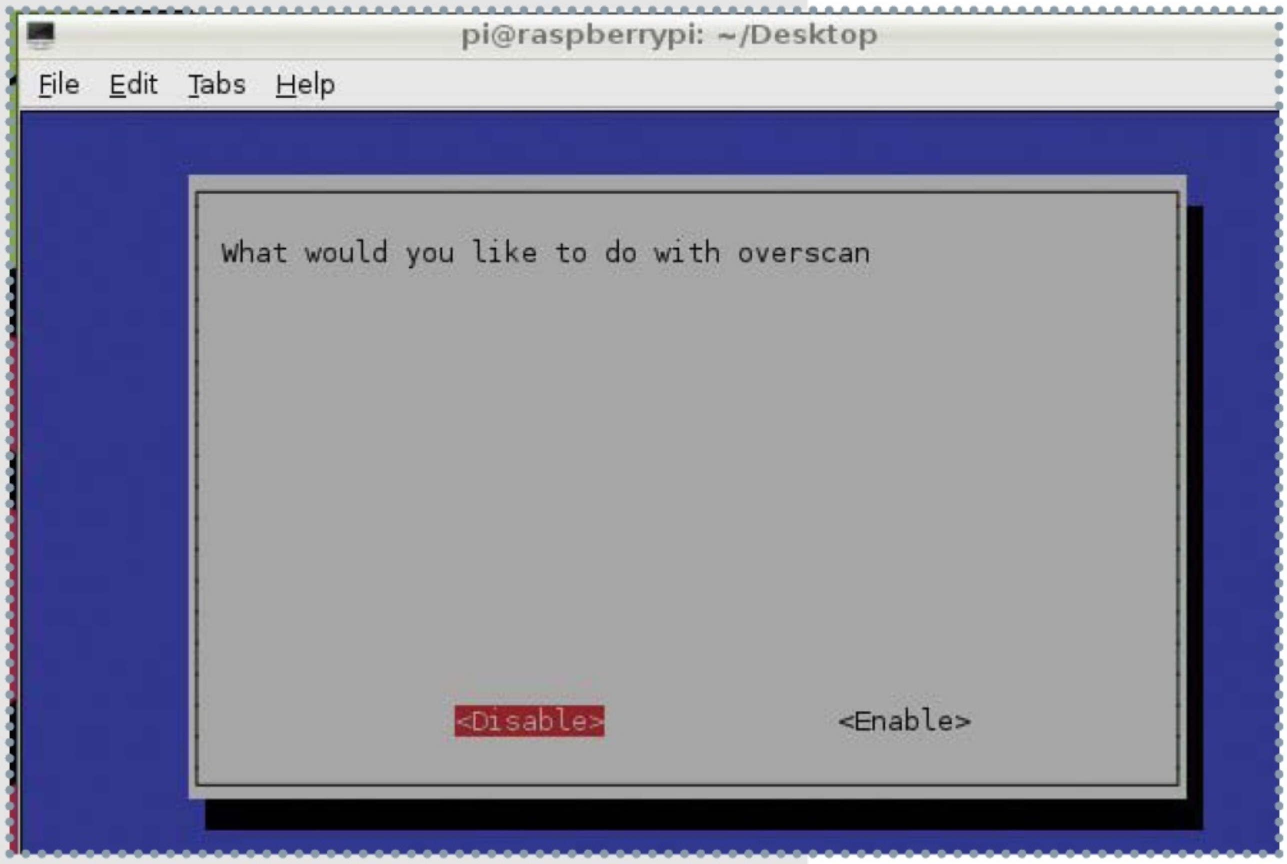
"The default user/password combination is pi/raspberry"

Problem 12: Startx fails to start window manager

Solution: If you just get errors instead of a working desktop when trying to use the startx command, you may be out of disk space. This possibly either means you need to expand the rootfs using the raspi-config tool if you have an SD card bigger than 2GB. If not, you probably need to get a bigger SD card. If you know there's space, try renaming the .Xauthority file under your home directory, temporarily.

Left Have you checked whether you are out of disk space?

Problem 13: The visible desktop goes off the screen



Solution: You probably want to turn overscan off. Use the raspi-config tool to disable overscan.

If you now see a big black border round the edges of your screen, you can usually use your TV's settings to zoom in.

If not, try setting the overscan manually in the config.txt using the properties overscan_left, overscan_right, overscan_top and overscan_bottom.

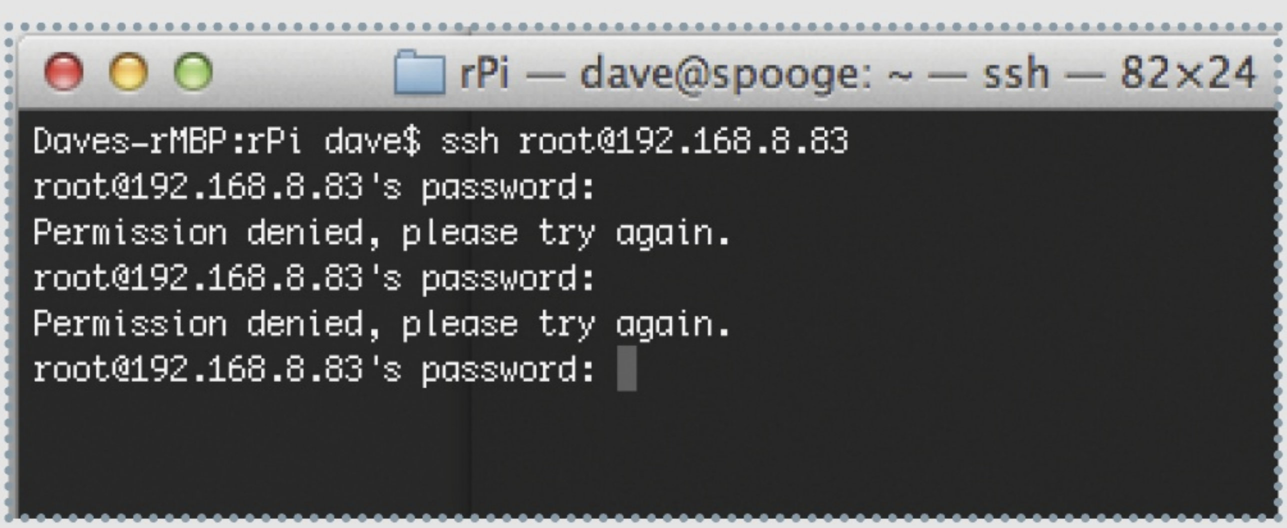
"If you now see a big black border round the edges of your screen, you can usually use your TV's settings to zoom in"

Problem 14: I don't know the root password

Solution: This is probably because the root account in many newer Linux distributions is disabled by default – and as such doesn't have a password. You can enable a password by entering the following command in the terminal:

```
sudo passwd root
```

Exercise caution when doing this, however, since meddling with root accounts can be dangerous.



Left Make sure you don't forget your new root password...

Problem 15: I can't install new software with apt-get

Solution: If you're seeing the error 'Package xx is unavailable' then you probably need to update the Apt tool first. Seeing this error means that your software list is out of date.

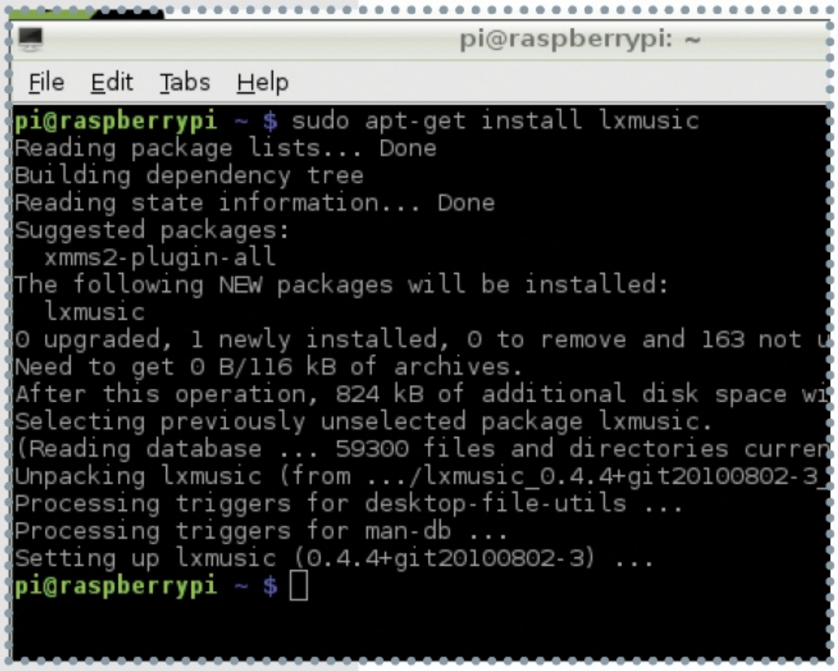
Start by running this command:

```
sudo apt-get update
```

Let it update its list of packages (you'll need a network connection) – then do:

```
sudo apt-get upgrade.
```

When this completes, try installing your package over again.



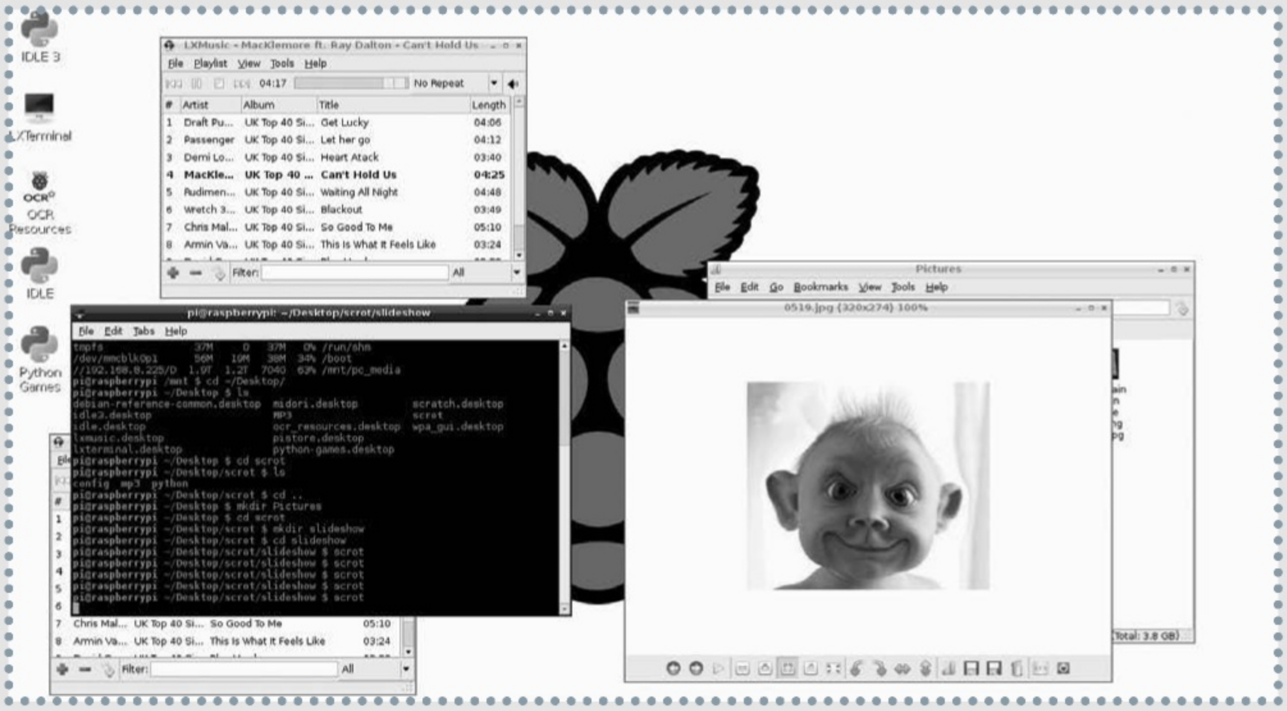
"The Pi's composite output defaults to NTSC (American). Some PAL (European) TVs may either now show and image, or display it in black and white"

Problem 16: Composite output is only black and white

Solution: The Pi's composite output defaults to NTSC (American). Some PAL (European) TVs may either now show an image, or display it in black and white. To solve this, change the config.txt on the SD card to add/modify:

```
sdtv_mode =x
```

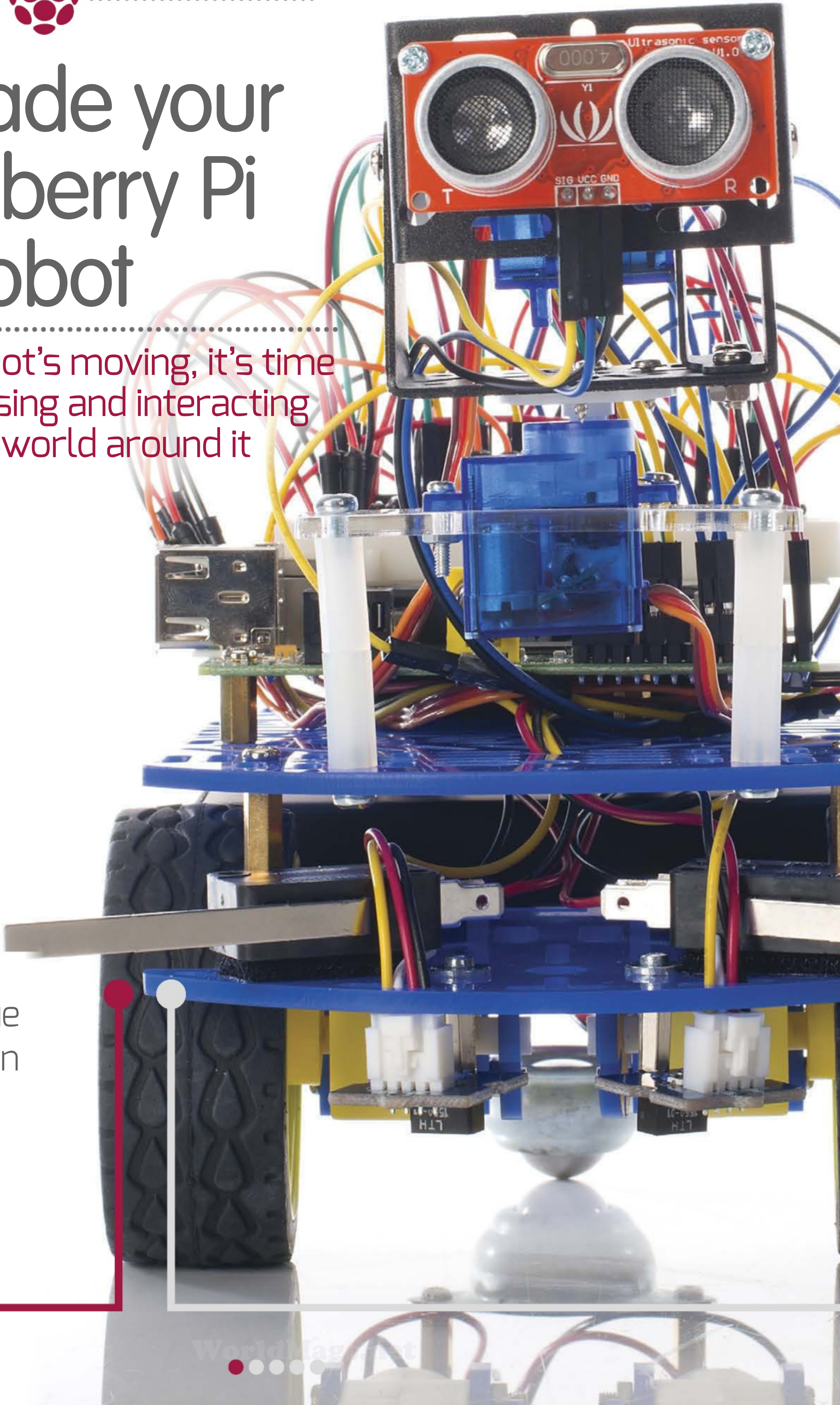
Where x is: 0 – NTSC; 1 – Japanese NTSC
2 – PAL; 3 – Brazilian PAL



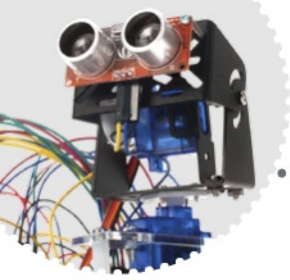
Left If your PI's gone greyscale then check config.txt on your SD

Upgrade your Raspberry Pi robot

Now your robot's moving, it's time to get it sensing and interacting with the world around it



"Using analogue sensors, we can build a robot that is capable of following or avoiding light"



Installing microswitches

Give your robot the sense of touch and train it to react when it bumps into something

In the previous issue of **RasPi**, we constructed our base robot. Now that we've got a robot that can move any way we want it to, let's move on to the simplest form of interaction: touch. It may not be as sophisticated as we experience as humans, but giving our robot its first sense will help it to navigate its own path, giving it a very basic form of intelligence.

Adding a sense of touch can be handled in quite a few different ways, but the quickest and easiest method is by adding some 'antennae' to your robot in the form of microswitches. Given their name, they aren't so much micro but they do have long arms that protrude, making them perfect for mounting on the front of your robot. If your switch hasn't got a lever or it isn't long enough, you can always try adding or extending it using a piece of dowel or a drinking straw.

Adding multiple switches gives our robot a greater sense of its surroundings and allows a very simple bit of code to control how it should operate. As it will be mostly moving forward, we will only need to add switches to the front. So let's begin by creating the circuit and testing it.

"The motor driver we will use is called an L293D, otherwise known as an H-Bridge"

Caution

While we've carefully constructed this feature with safety in mind, accidents can happen. Imagine Publishing cannot be held responsible for damage caused to Raspberry Pis and associated hardware by following this feature.

THE PROJECT ESSENTIALS

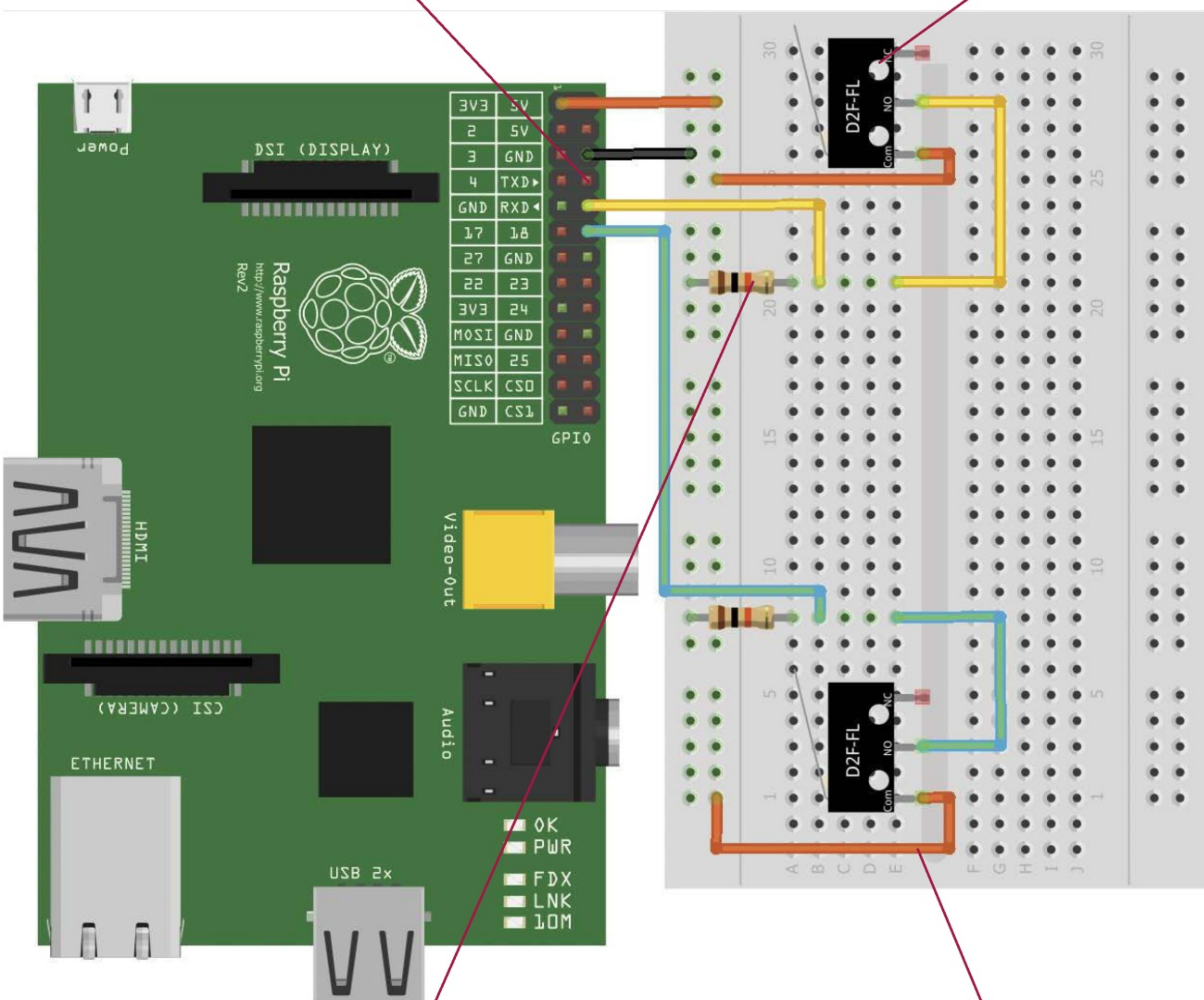
Breadboard
Jumper wires
2x 10K resistors
2x CPC microswitches

NO/NC

Most switches are labelled with NO and NC; which stands for Normally Open and Normally Closed. Open simply means no current can pass through

Additional switches

You can easily add more switches, not just bumpers, by following the circuit and connecting it to a free pin

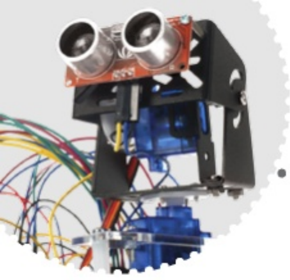


Pull-down Resistors

As we are dealing with digital logic, the switch has to be either on or off, and the resistor helps this by weakly pulling the pin low

3v3 Power

As the Raspberry Pi has no protection from overvoltage, we can't input more than 3.3V otherwise we risk frying the processor



Testing your microswitches

Now the switches are wired up, let's get them working

Wiring them up is nice and simple, but as mentioned, it is important to remember that the Raspberry Pi is only 3.3V tolerant when using inputs, so we are only going to use the 3V3 pin and not the 5V pin.

The Python code to read inputs is nice and straightforward. Since we have one switch per GPIO pin, we just get Python to tell us what state it is in when we ask. So the first thing we will do is import our usual libraries and then set the pins to BCM board mode. In GPIO.setup we are going to tell Python to set pins 15 and 18 as inputs.

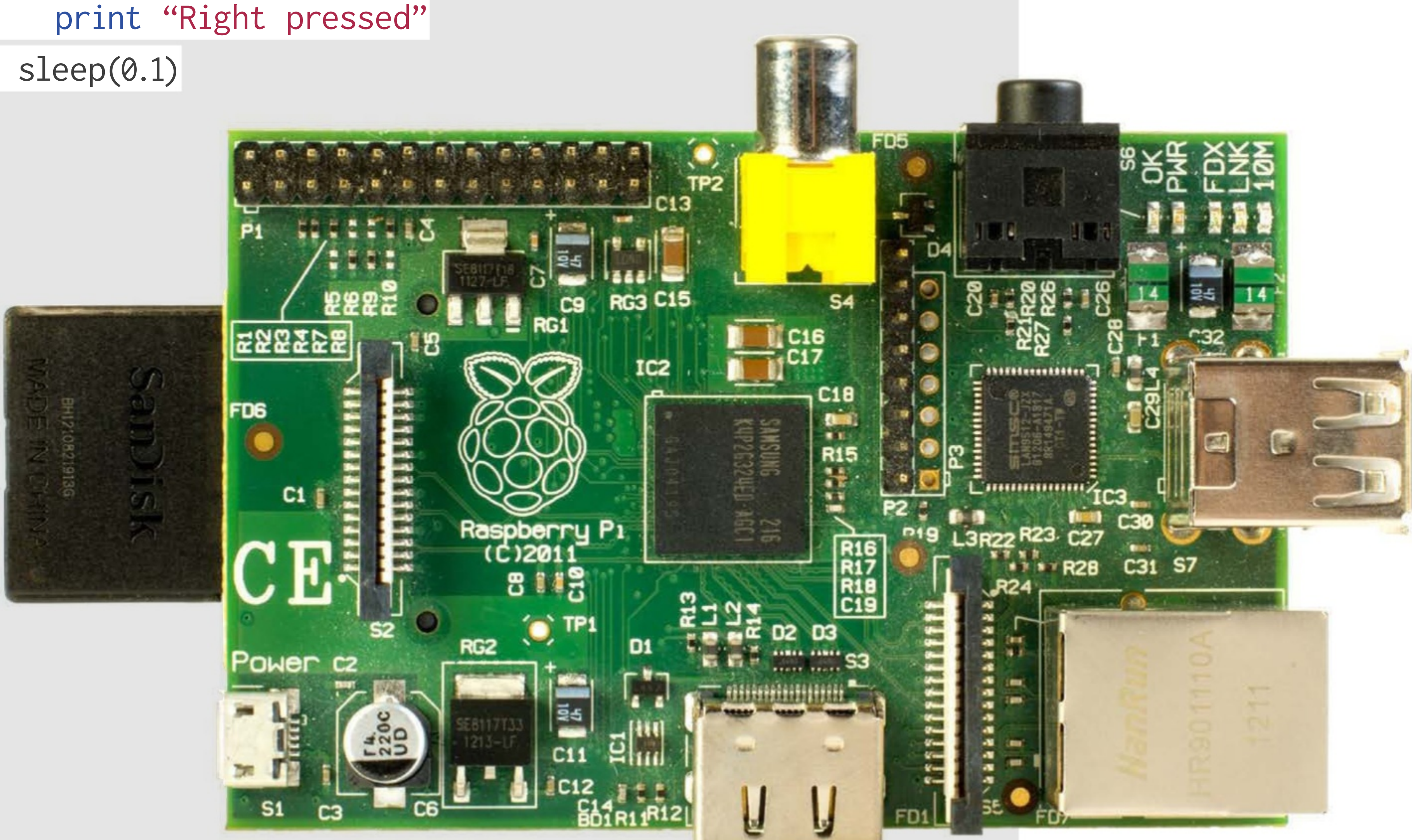
Creating a while True: loop will create an infinite loop, as the condition is always true. While in the loop, we shall store the current state of the input into a variable, and then use an if statement to check if it is a 1 for pressed or a 0 for not pressed. All we are going to do is display on the screen which switch has been pressed; it will also help us work out on which side to place the microswitch.

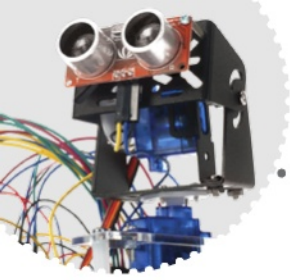
"The Python code to read inputs is nice and straightforward. Since we have one switch per GPIO pin, we just get Python to tell us what state it is in when we ask"

The Code TESTING THE MICROSWITCHES

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)


GPIO.setup(18, GPIO.IN)
GPIO.setup(15, GPIO.IN)


while True:
  inputleft = GPIO.input(18)
  inputright = GPIO.input(15)
  if inputleft:
    print "Left pressed"
  if inputright:
    print "Right pressed"
  sleep(0.1)
```

"All we are going to do is display on the screen which switch has been pressed"





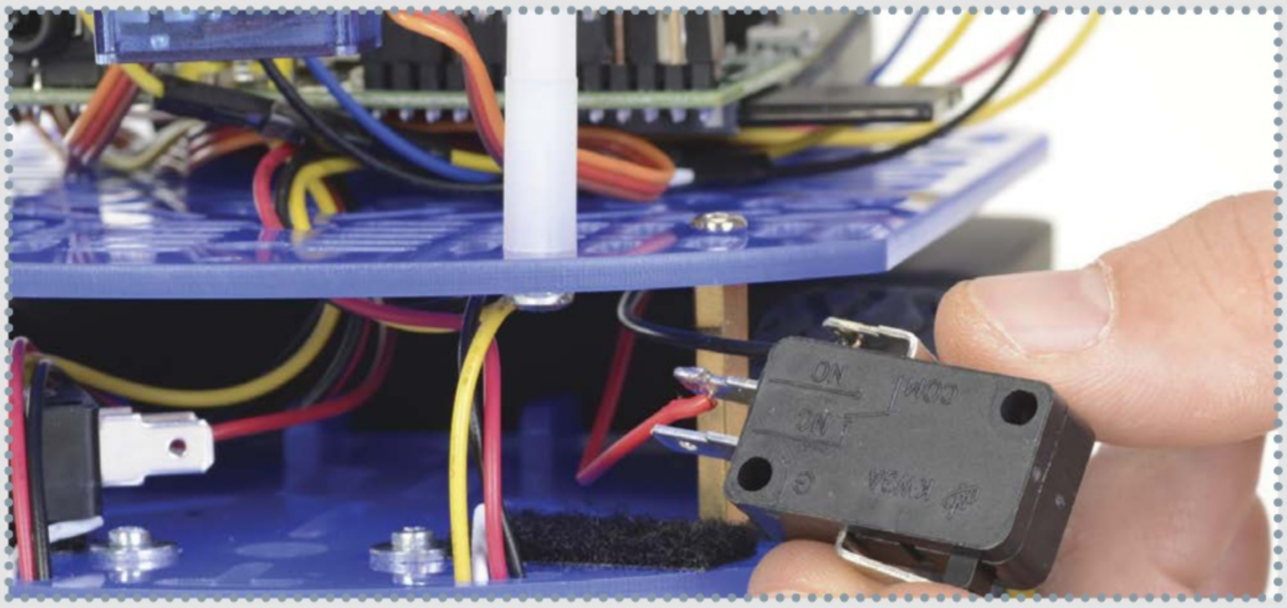
Completing the 'bumping' robot

It's time to add the switches to the robot and find some walls to test it with

We'll mount the switches to the front of the robot, fixing it down with double-sided tape or Blu-tack so the levers can stick out enough to be pressed when it touches an object. Re-using the motor function code we created before, we can easily add the microswitch support. So this time if an object presses the left microswitch, we tell the motors to switch into reverse for a second then stop. Hopefully this is long enough to move the robot away from the object so we can now turn just the left-hand motor on for 2 seconds before continuing on its new path. This is a big step – we're implementing AI and making the robot smart.

Variations can be made to refine our robot, such as creating a reverse for the right-hand motor and having it spin on the spot to create a new path to continue on.

"If an object presses the left microswitch, we tell the motors to switch into reverse for a second then stop. Hopefully this is long enough to move the robot away from the object"

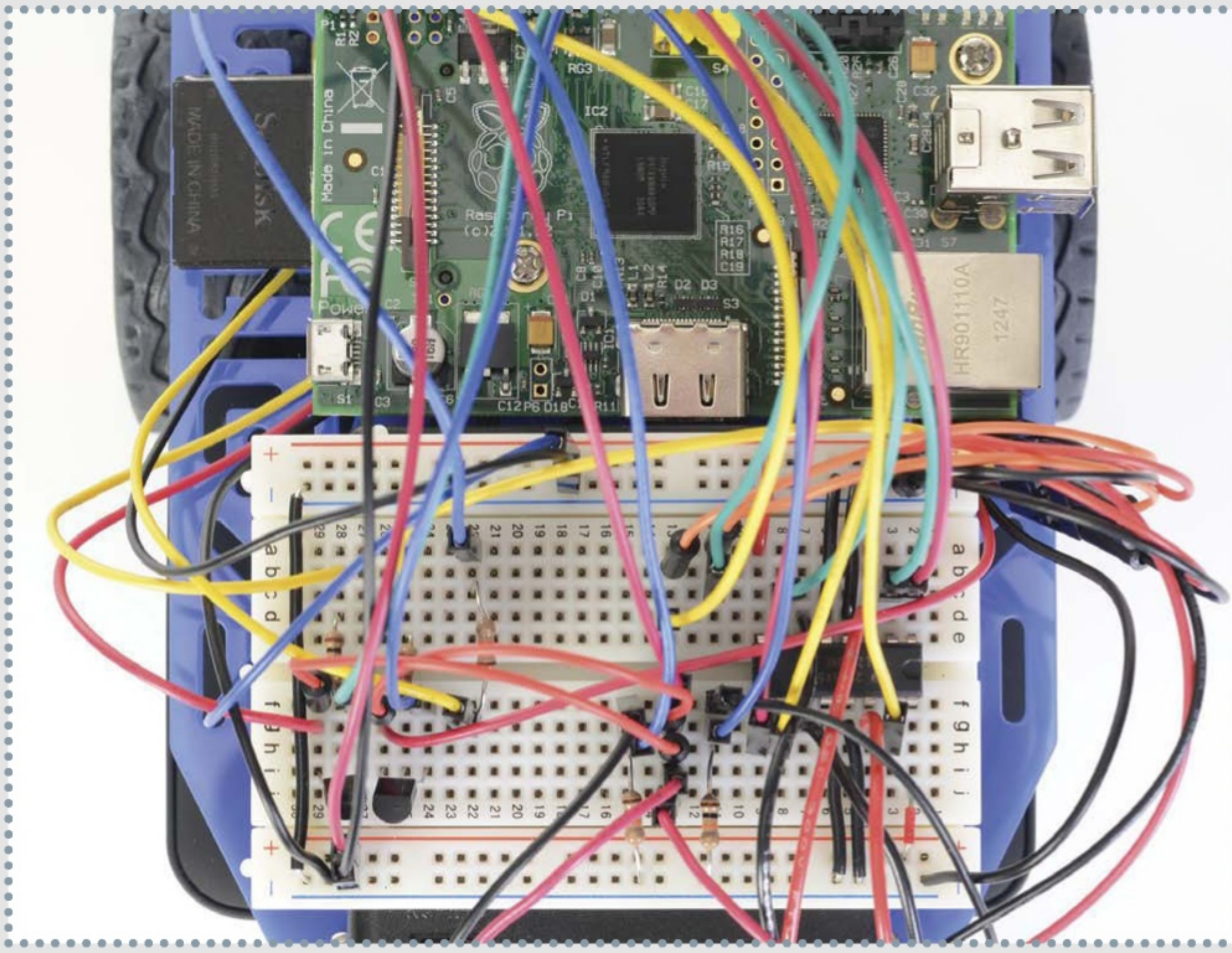


01 Mount the switch

Try to place the microswitches as close to the front of the robot as possible, spaced far enough apart so we can work out what direction the robot is facing when it hits something.

02 Wire it up with the motor circuit

Finding a couple of spare tracks (vertical columns) on the breadboard, add the GPIO jumper cable to the pull-down resistor and connect the switch as shown in the diagram.



Above left Placing the microswitches takes a little trial and error

Below left Swipe back a few pages to see the Frizting diagram for this

"Mount the switches to the front of the robot, fixing it down with double-sided tape or Blu-tack so the levers can stick out"

03 Log in with SSH

As our robot will be starting to run around freely, it is a good idea to provide the Raspberry Pi with its own battery. Using Wi-Fi, we can remotely connect using SSH to emulate the Pi's terminal.

04 Create and save your work

As before with the motors, we shall create our script using nano. Let's do this by typing

`nano bumpers.py`

Saving different scripts allows the testing of individual parts. We can also use them as a reference for creating bigger scripts.

05 Test it in situ

Copying the example script into `bumpers.py`, followed by `Ctrl+X` with a `Y` to save, we can test it out and make any hardware modifications. With the script running, press a microswitch and see what happens!

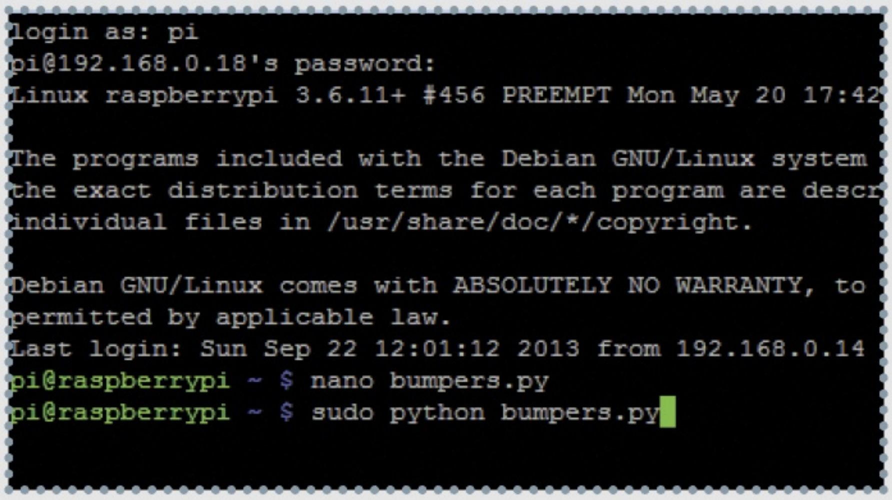
06 Modify and improve your code

When you start the script, the motors will start turning forward. Pressing a switch should then reverse the motors and spin one motor before going forward again. Play with the variables and tweak its response to your preference.

> "When you start the script, the motors will start turning forward. Pressing a switch should then reverse the motors and spin one motor before going forward again"

Below left The script you need to copy is on the next page

Below right We're re-using chunks of the movement code here



```
login as: pi
pi@192.168.0.18's password:
Linux raspberrypi 3.6.11+ #456 PREEMPT Mon May 20 17:42

The programs included with the Debian GNU/Linux system
the exact distribution terms for each program are descr
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to
permitted by applicable law.
Last login: Sun Sep 22 12:01:12 2013 from 192.168.0.14
pi@raspberrypi ~ $ nano bumpers.py
pi@raspberrypi ~ $ sudo python bumpers.py
```

```
while True:
  inputleft = GPIO.input(18)
  inputright = GPIO.input(15)
  if inputleft:
    print "Left pressed"
    backward(70)
    sleep(1)
    stop()
    left(50)
    sleep(2)
  elif inputright:
    print "Right pressed"
    backward(70)
    sleep(1)
    stop()
```

The Code BUMPING ROBOT

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)


GPIO.setup(18, GPIO.IN)
GPIO.setup(15, GPIO.IN)

GPIO.setup(24,GPIO.OUT)
GPIO.setup(23,GPIO.OUT)
GPIO.setup(25,GPIO.OUT)
GPIO.setup(9,GPIO.OUT)
GPIO.setup(10,GPIO.OUT)
GPIO.setup(11,GPIO.OUT)

Motor1 = GPIO.PWM(25, 50)
Motor1.start(0)
Motor2 = GPIO.PWM(11, 50)
Motor2.start(0)

def forward(speed):
  GPIO.output(24,GPIO.HIGH)
  GPIO.output(23,GPIO.LOW)
  GPIO.output(9,GPIO.HIGH)
  GPIO.output(10,GPIO.LOW)
  Motor1.ChangeDutyCycle(speed)
  Motor2.ChangeDutyCycle(speed)

def backward(speed):
  GPIO.output(24,GPIO.LOW)
  GPIO.output(23,GPIO.HIGH)
  GPIO.output(9,GPIO.LOW)
```

Don't fry the Pi!

It is important to check the specifications of any sensor to make sure it is compatible with 3.3V power supply.

"Variations can be made to refine our robot, such as creating a reverse for the right-hand motor and having it spin on the spot to create a new path"

The Code BUMPING ROBOT

```
  GPIO.output(10,GPIO.HIGH)
  Motor1.ChangeDutyCycle(speed)
  Motor2.ChangeDutyCycle(speed)

def left(speed):
  GPIO.output(24,GPIO.HIGH)
  GPIO.output(23,GPIO.LOW)
  Motor1.ChangeDutyCycle(speed)

def right(speed):
  GPIO.output(9,GPIO.HIGH)
  GPIO.output(10,GPIO.LOW)
  Motor2.ChangeDutyCycle(speed)

def stop():
  Motor1.ChangeDutyCycle(0)
  Motor2.ChangeDutyCycle(0)


while True:
  inputleft = GPIO.input(18)
  inputright = GPIO.input(15)
  if inputleft:
    print “Left pressed”
    backward(100)
    sleep(1)
    stop()
    left(75)
    sleep(2)
  elif inputright:
    print “Right pressed”
    backward(100)
    sleep(1)
```

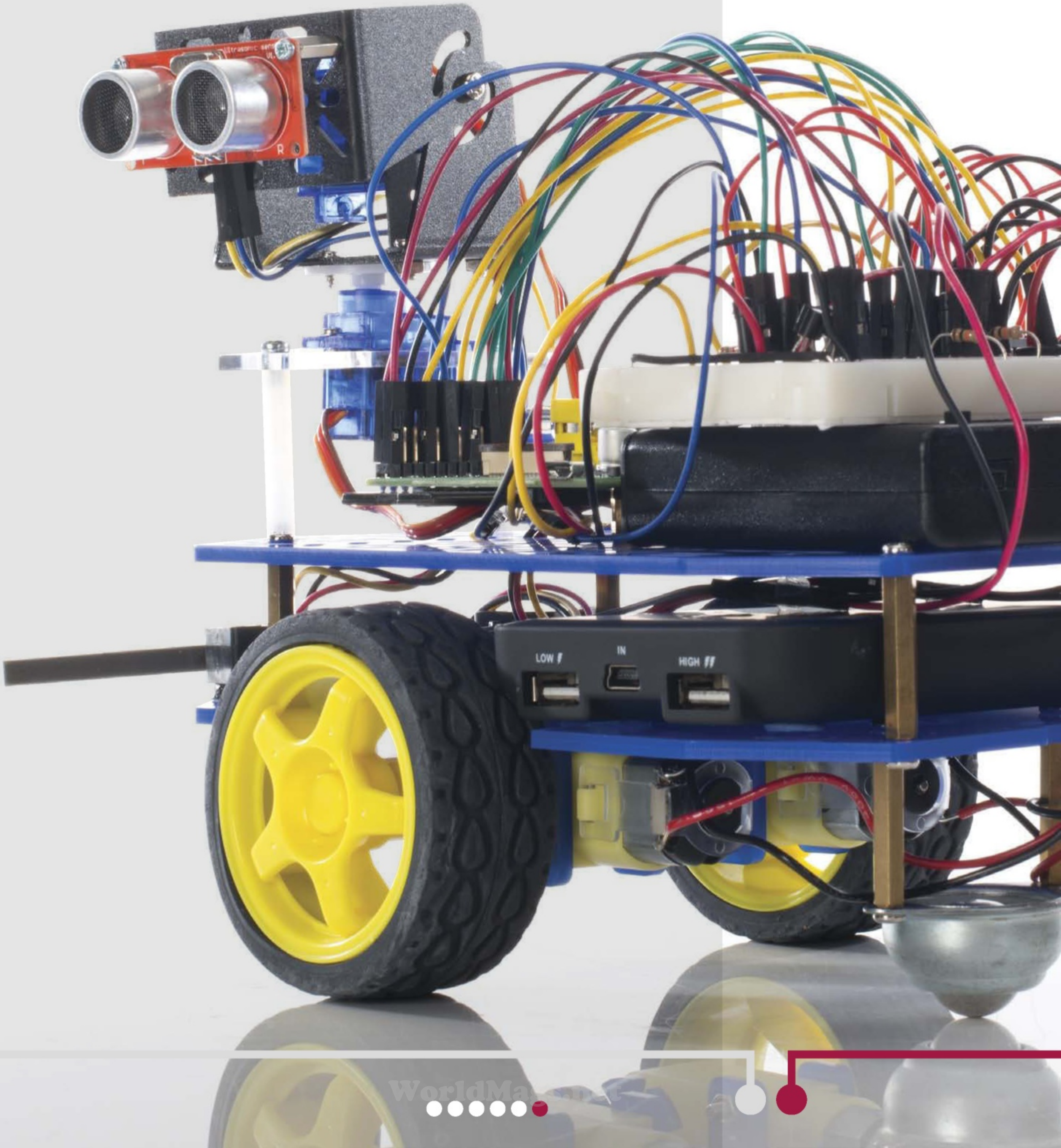
Digital switches

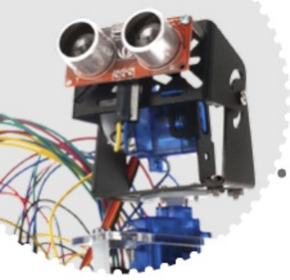
A switch is a perfect digital signal, as it can only be one of two states: on or off.

"Creating a while True: loop will create an infinite loop, as the condition is always true"

The Code BUMPING ROBOT

```
        stop()
        right(75)
        sleep(2)
    else:
        forward(75)
    sleep(0.1)
```





Line-following sensors

Give your robot a track to follow using masking tape or inked paper

So far the robot can decide its own path, which is a great thing for it to do, but it could end up in trouble. Let's help it follow a set path.

One solution is to add some line sensors to the underside so we are able to control it by using some masking tape on a dark floor (or some dark tape on a light floor). This can be used in a number of different ways.

By marking a line on the floor, we can get the robot to follow it obediently; even by throwing in a few curves, it should be able to navigate a set path. Or it is possible to tackle it another way by adding a perimeter around the robot, allowing us to restrict the robot to a box or set area.

Line following is best achieved with two-wheeled robots as their ability to quickly change direction is important. The principal is that as a sensor is triggered we can stop a corresponding motor, allowing the robot to swing around to stay on the line.

> "Add some line sensors to the underside so we are able to control it by using some masking tape on a dark floor (or some dark tape on a light floor)"

THE PROJECT ESSENTIALS

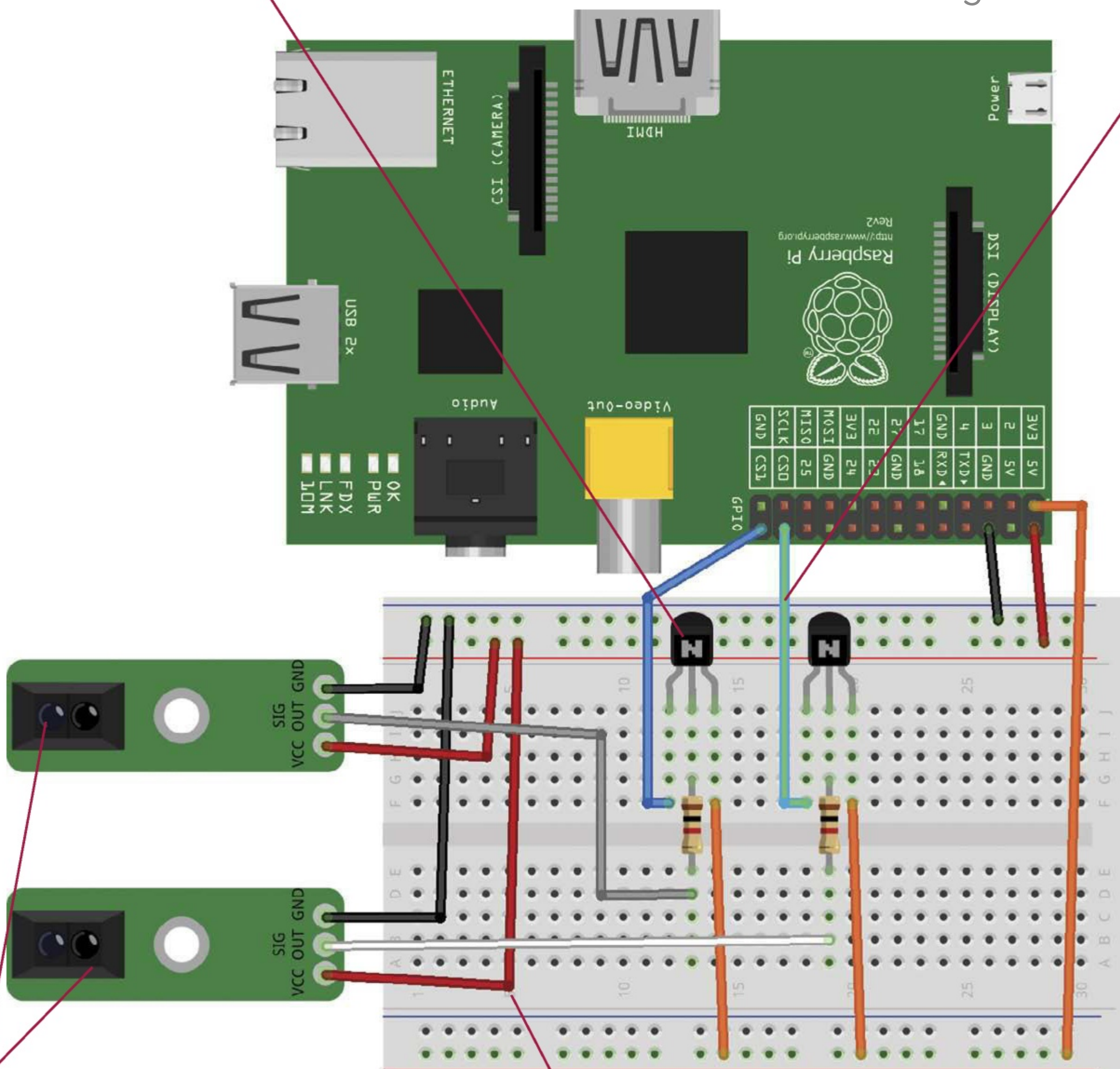
Breadboard
Jumper cables
2x 2N3904 transistors
2x 1K resistors
2x Line detector sensors

Lower the current

The transistors only need a small amount of current to actually work; a resistor helps to smooth out the sensors' output

Safety first

Thanks to the transistor, we have a much safer voltage going back into the GPIO pins when using 5-volt electronics



Line sensors

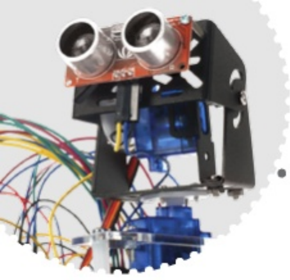
Sensors come in a variety of shapes and sizes, but most have a common set of pins; the important one is the OUT pin

Power

Most sensors are only available in 5-volt form; we need a transistor to switch the voltage to a Raspberry Pi-safe level

Making voltage safer

Transistors work just like a switch, being able to turn power on and off. Using it to switch the 3.3V power to the GPIO is a much safer method



Testing the line sensors

Now you've given your robot eyes, let's try them out

With the line sensors wired up and the Raspberry Pi switched on, we can now test them. This Python script is very similar to the microswitch test code because we are just reading the GPIO pin's status, checking if it is high (a 1 or on) or if it is low (0 or off).

As some sensors work differently to others, we need help to understand the output. Displaying the current sensor data on the screen allows us to work out how the sensor responds on black and white surfaces and plan the code accordingly.

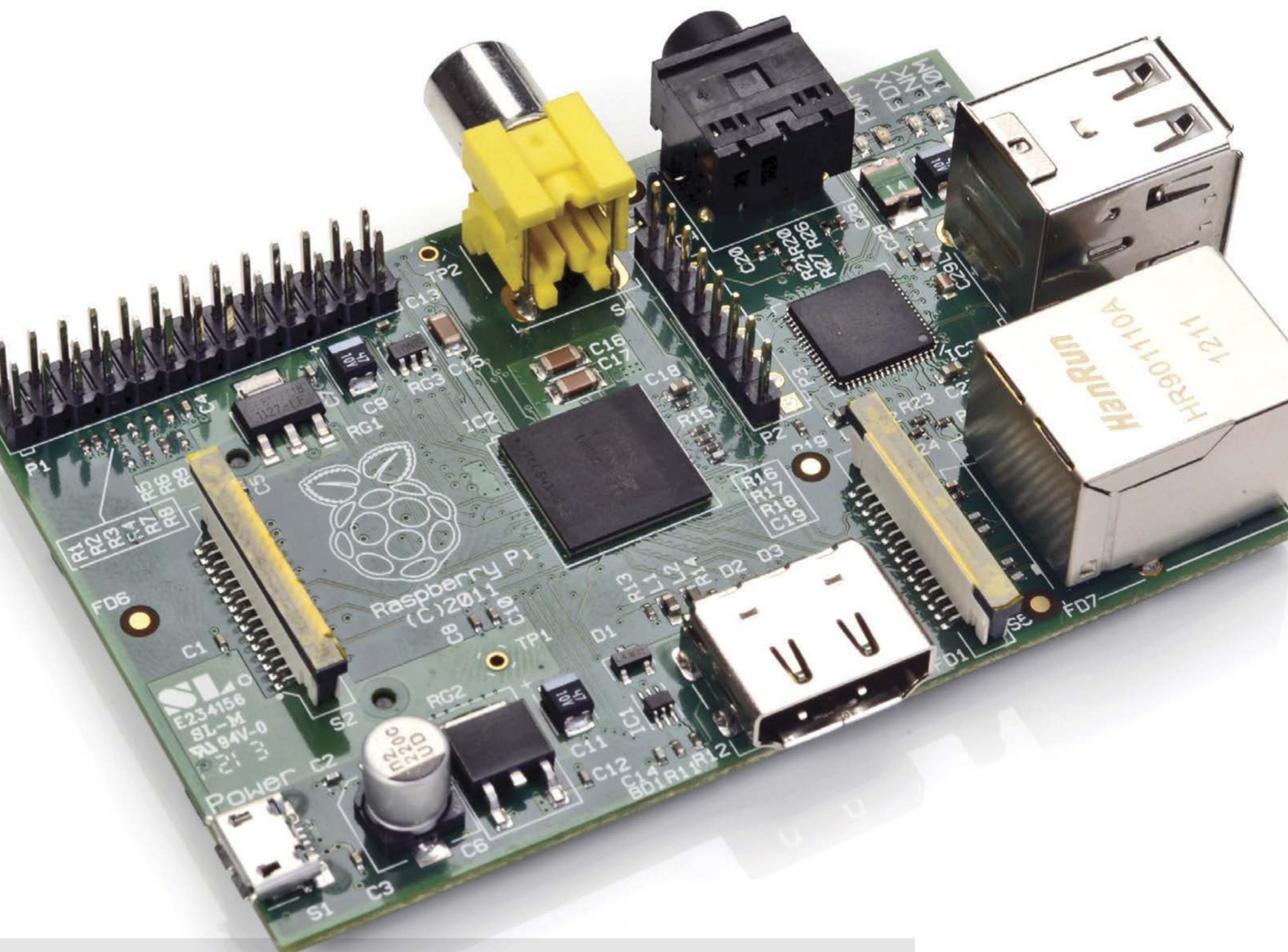
01 Start your project

Log in via SSH or start a terminal on your Raspberry Pi and create the linefollow.py script with `nano linefollow.py`. This will be our test script for the finished line-following robot.

02 Read the sensors

Copy the test script into the file. As each sensor is slightly different, we may need to tweak the code slightly to suit, so test what you have and interpret the output.

"This Python script is very similar to the microswitch test code because we are just reading the GPIO pin's status"



03 Print to screen

Save the file as before. You'll notice the code we've supplied has print statements to show if the sensor is picking up any difference between light surfaces and dark surfaces.

04 We have data

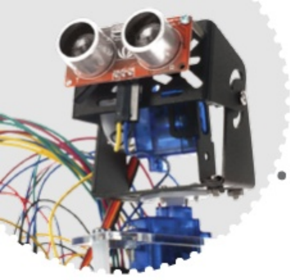
If everything is wired up correctly, the screen will start filling up with sensor data, letting us know if it can see black or white. Put some paper in front of the sensor to try it out.

> "Displaying the current sensor data on the screen allows us to work out how the sensor responds on black and white surfaces"

The Code TESTING THE LINE SENSORS

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)

Input1 = 7
Input2 = 8

GPIO.setup(Input1,GPIO.IN)
GPIO.setup(Input2,GPIO.IN)


while True:
   Sensor1 = GPIO.input(Input1)
   Sensor2 = GPIO.input(Input2)

   if Sensor1 == GPIO.HIGH:
      print "Sensor 1 is on White"
   else:
      print "Sensor 1 is on Black"

   if Sensor2 == GPIO.HIGH:
      print "Sensor 2 is on White"
   else:
      print "Sensor 2 is on Black"

   print "------"
   sleep(1)


GPIO.cleanup()
```

"If everything is wired up correctly, the screen will start filling up with sensor data, letting us know if it can see black or white"



Finalise your line-following bot

It can see! Now put its new eyes to good use...

By now we should be used to controlling the motors, so building on that knowledge we can start to concentrate on the sensors. Most line followers use the same convention as microswitches, giving a high output to signal the sensor is over a black surface and a low output (or off) to signal that it's over a white surface.

When using a white masking tape line, we want the motor to stop when the sensor is touching the line, giving the other side a chance to turn the robot in order to correct its position.

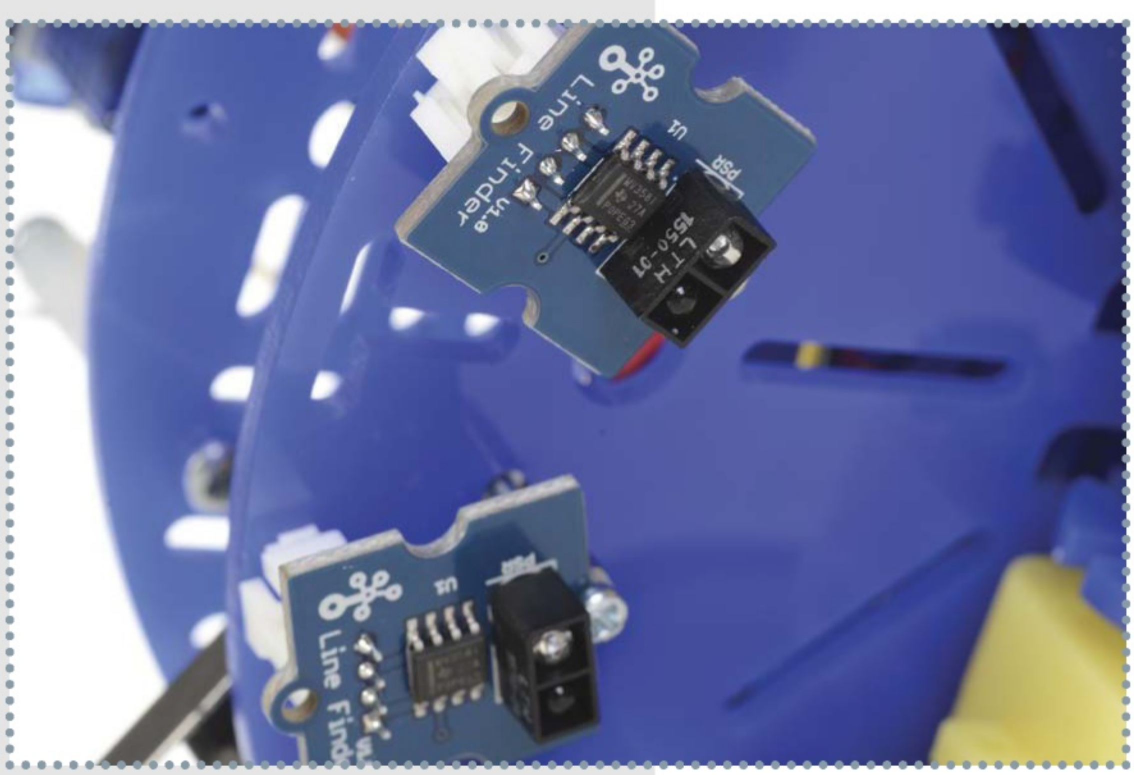
The code is nice and simple, so it can be easily modified to suit your own situation.

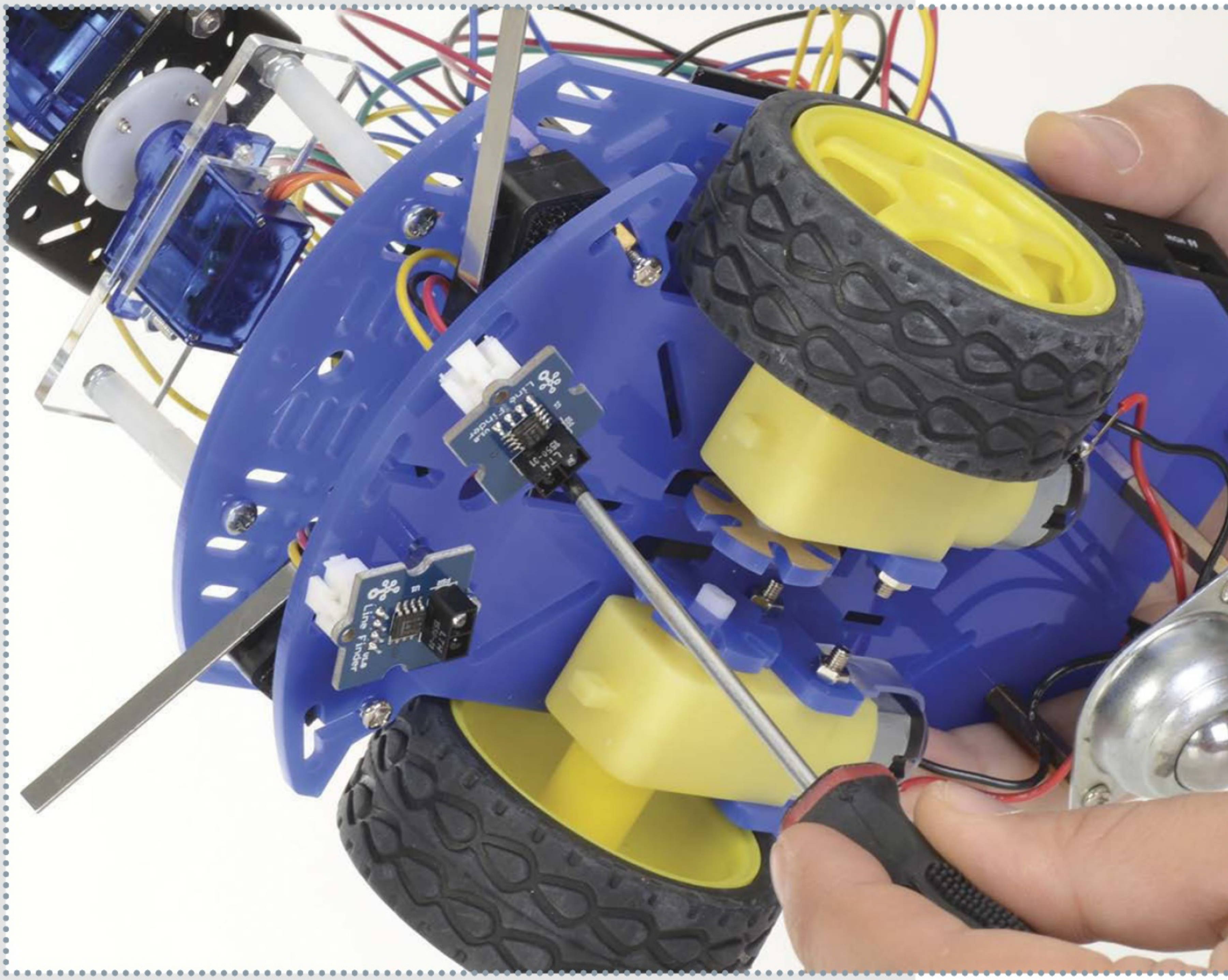
Big business

Some manufacturing plants use lines to guide robots around warehouses in an identical way to our robot, but on a much larger scale.

01 Mount the sensor

Using the hexagonal mounting rods, mount the sensors at about 10mm to cope with uneven floors. Most sensors will be sensitive enough at that distance; if not, there will be a potentiometer to adjust the sensitivity.





Above Fix the sensors securely, so they don't get knocked off

02 Adding to your breadboard

There should be plenty of room on your robot's breadboard, but make sure you use all the available 'tracks'. Keep your different types of sensors in their own little areas – get to know them so you can debug the hardware easily.

03 Add the sensor circuit

Place the two transistors and resistors on the breadboard, checking each pin is in its own column. Add the jumper cables from the sensors and power lines, and then to the GPIO pins.

04 Power up and log on

Connect the batteries for the motors and add power to the Raspberry Pi. Now log in using SSH on your computer so we are able to create our motor-controlled line sensor code.

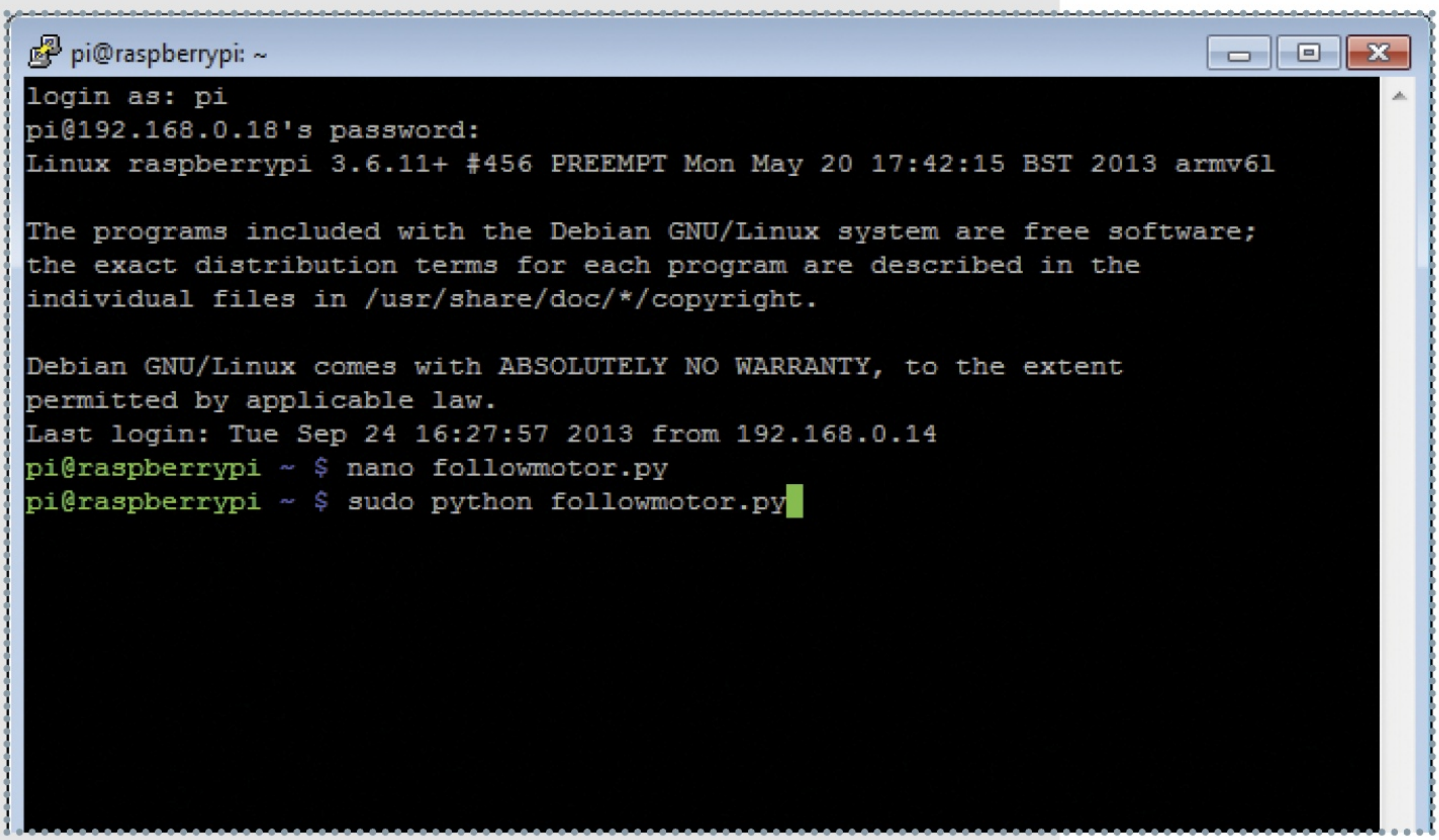
05 Creating the script

As you can see on the next page, the code for the line-following robot is similar to our previous code. While True: ensures the code loops until we stop it, and we've kept print statements in for debugging.

06 Testing your new script

All being well, your robot will now scoot off and find a line to follow. There are plenty of ways to improve and add to this code to make the bot's movements along the line smoother. It's also quite trivial to build this into your existing code.

Below Nano the new file and add the code from the next page



The Code BUMPING ROBOT

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)


GPIO.setup(7, GPIO.IN)
GPIO.setup(8, GPIO.IN)

GPIO.setup(24,GPIO.OUT)
GPIO.setup(23,GPIO.OUT)
GPIO.setup(25,GPIO.OUT)
GPIO.setup(9,GPIO.OUT)
GPIO.setup(10,GPIO.OUT)
GPIO.setup(11,GPIO.OUT)

Motor1 = GPIO.PWM(25, 50)
Motor1.start(0)
Motor2 = GPIO.PWM(11, 50)
Motor2.start(0)

def forward(speed):
    GPIO.output(24,GPIO.HIGH)
    GPIO.output(23,GPIO.LOW)
    GPIO.output(9,GPIO.HIGH)
    GPIO.output(10,GPIO.LOW)
    Motor1.ChangeDutyCycle(speed)
    Motor2.ChangeDutyCycle(speed)

def backward(speed):
    GPIO.output(24,GPIO.LOW)
    GPIO.output(23,GPIO.HIGH)
    GPIO.output(9,GPIO.LOW)
```

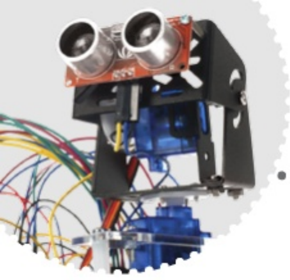
Sudo Python?

Prefix with sudo to elevate a program's permission level to a superuser. It's required to control the GPIO pins from Python, so don't forget it!

The Code BUMPING ROBOT

```
    GPIO.output(10,GPIO.HIGH)
    Motor1.ChangeDutyCycle(speed)
    Motor2.ChangeDutyCycle(speed)

def left(speed):
    GPIO.output(24,GPIO.HIGH)
    GPIO.output(23,GPIO.LOW)
    Motor1.ChangeDutyCycle(speed)

def right(speed):
    GPIO.output(9,GPIO.HIGH)
    GPIO.output(10,GPIO.LOW)
    Motor2.ChangeDutyCycle(speed)

def stop():
    Motor1.ChangeDutyCycle(0)
    Motor2.ChangeDutyCycle(0)


while True:
    sensor1 = GPIO.input(7)
    sensor2 = GPIO.input(8)
    if sensor1 == GPIO.LOW:
        print “Sensor 1 is on white”
        stop()
    else:
        left(60)
    if sensor2 == GPIO.LOW:
        print “Sensor 2 is on white”
        stop()
    else:
        right(60)
    sleep(0.05)
```

“The code is similar to our previous code. While True: ensures the code loops until we stop it”



Ultrasonic sensing

Now that the robot can touch, and it can see beneath its wheels, what about a new type of sight?

Let's start making things a little more complicated by adding an ultrasonic sensor and placing it onto a pan-and-tilt mounting. Ultrasonic sensors are used in different ways to judge distance by emitting an ultrasonic pulse and counting how long it takes to bounce off an object then back to the receiver. Cars that come with reverse parking sensors work in the same way to give an audible tone depending on how far away an object is.

Using an ultrasonic sensor on your robot will give it a chance to take action as it approaches an object such as a wall, with enough time to evaluate and choose a new path.

Ultrasonic sensors come in two varieties, based on the number of pins. Both types work in a very similar way. Since we would like to use the same Python code for both, we would wire the 4-pin sensor to act like a 3-pin ultrasonic sensor. However, we will focus on the affordable 3-pin model from Dawn Robotics.

As we only require one GPIO pin, we will first need to set it as an output and send a 10ms pulse to trigger the sensor to start and begin counting. Next we switch to an input to wait for the pin to go high, at which point we stop timing and calculate how long that took. The last thing needed is to convert the time in sound into a measurement we can read, in this case centimetres.

THE PROJECT ESSENTIALS

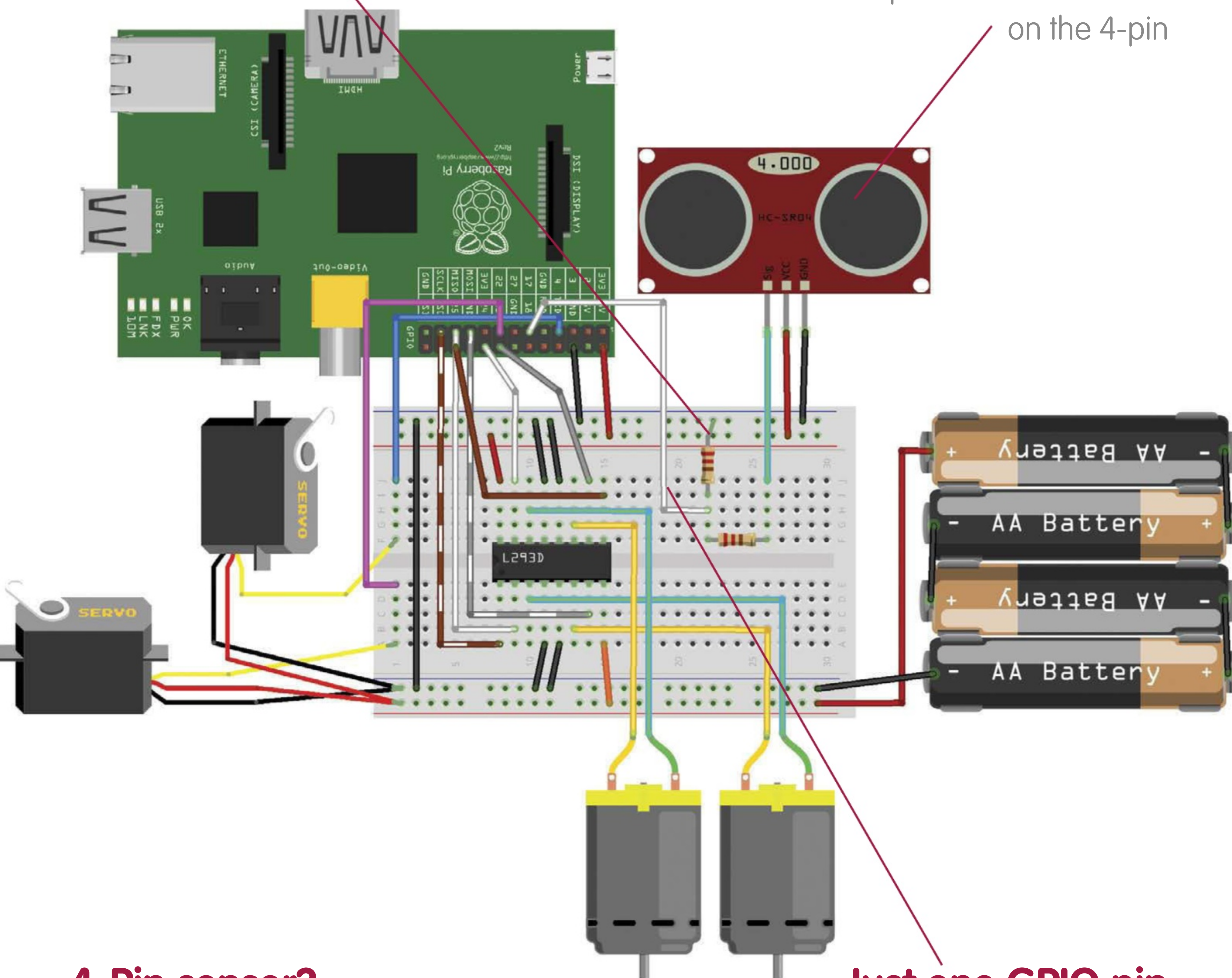
Dagu pan & tilt kit
Jumper cables
Seeed Studio ultrasonic sensor
2x 2K2 resistors
1x 10K resistor

Voltage divider

As again we are dealing with 5-volt sensors, we need to lower the voltage to 3.3 volts to make it safer for use with the Raspberry Pi

3-Pin sensor

We're using a 3-pin sensor which has a combined Echo/Trig pin. The functions perform the same as on the 4-pin

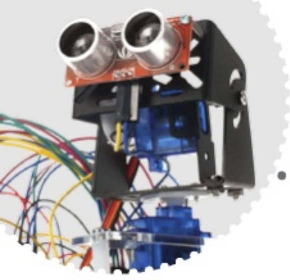


4-Pin sensor?

The most common is the 4-pin HC-SR04, capable of calculating distances up to 4 metres. Aside from the power and ground, it contains Trig and Echo pins

Just one GPIO pin

To save on GPIO pins, one pin will switch quickly between output and input to send and receive a pulse. This will work with 4-pin models too



Add a pan-and-tilt kit

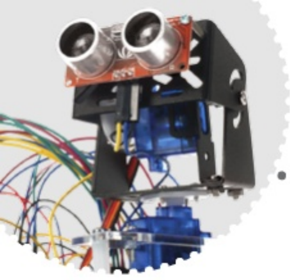
Wouldn't it be great to take readings from different angles? Here's how...

Pan-and-tilt mounts are very useful since they can be combined with any sort of sensor, giving the robot an ability to 'move its head' around and sense what is around it without physically moving its body. The panning and tilting is controlled by two special motors called servos.

Servos allow very precise movement within their range, typically between 0 and 180 degrees. They do this by using some very precise timing to send a pulse. The time between the pulses tells the servo its angle.

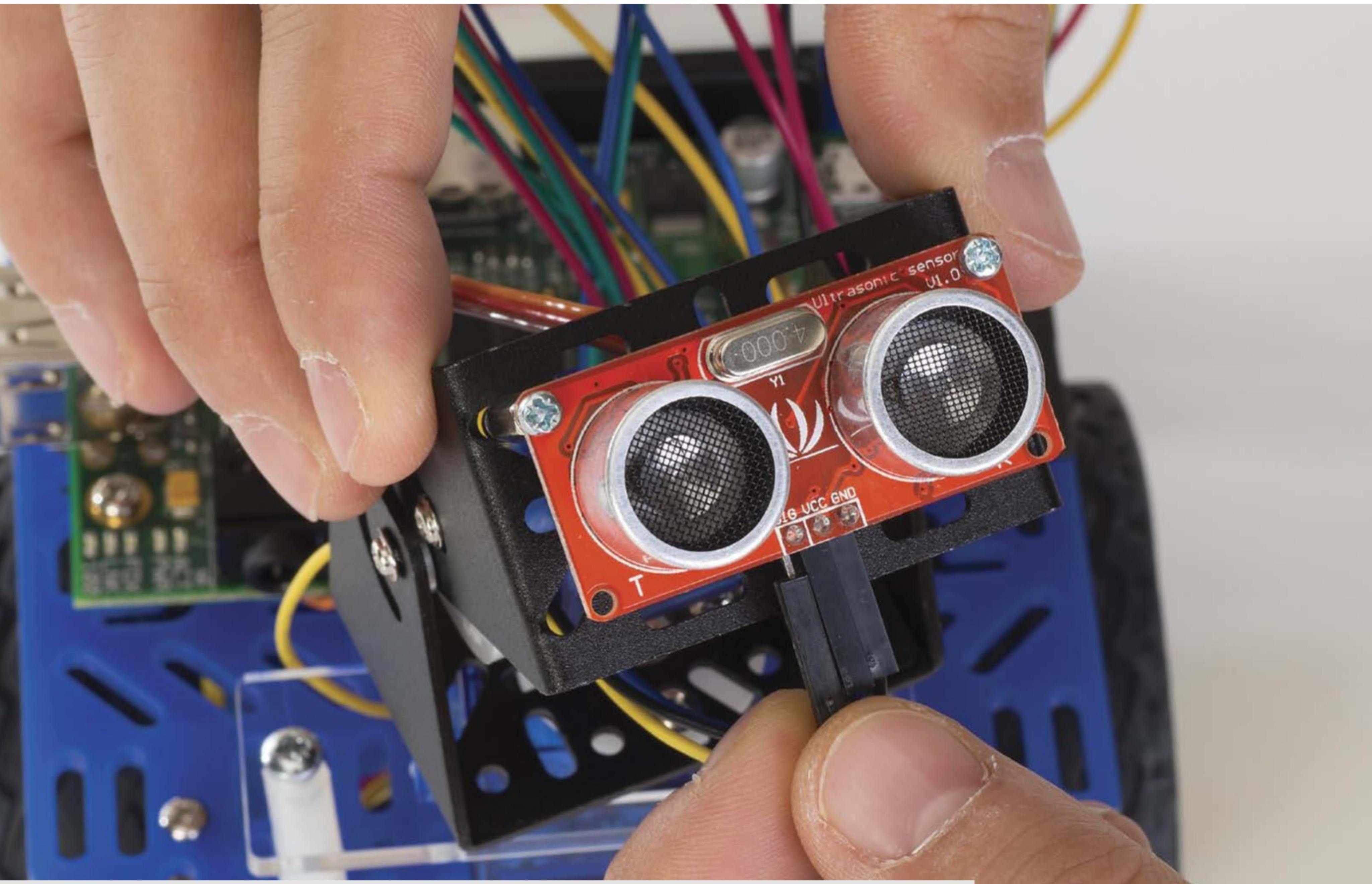
Typically the Raspberry Pi, being a not-so-great real-time device, would sometimes struggle maintaining a steady pulse, as it could forget what it was doing and go off to check some emails, for instance. Therefore Richard Hirst wrote a kernel for Linux called ServoBlaster, which handles the timing required perfectly, regardless of how much is happening. The kernel takes control of some of the timing registers to provide an accurate clock. All that is required is to send the angle you need to /dev/servoblaster and the servo will spring to life!

"Servos allow very precise movement within their range, typically between 0 and 180 degrees. They do this by using some very precise timing to send a pulse"



Installing your pan & tilt

It's a fiddly job, but well worth the trouble



Now we've taken care of the circuit, let's set them up; first we need to get the kernel so let's download that now. Type the following into your RasPi terminal:

```
wget https://github.com/Boeeerb/LinuxUser/raw/master/servod
```

And make it executable:

```
chmod +x servod
```

Lastly, run it – remember, every time you reboot or switch on your Pi, you will just need to type this line:

```
sudo ./servod
```

The pre-compiled kernel is already configured to use pins 4 and 22 as servos, so let's hook everything up.

The servos will need to be powered separately as they are at heart just motors with a little bit of circuitry.

The code we have (listed just after this section) will combine the wheel motors, servos and ultrasonic. The end result will involve the robot moving forward until it senses an object less than 30cm away, stop, then turn the pan-and-tilt left, check the distance, then turn the ultrasonic on the pan-and-tilt right and pick whichever direction has a further distance until the next object.

01 Assemble the kit

The pan-and-tilt mount allows a full view of 180 degrees from left to right, up and down – great for adding ultrasonics or even a camera. The servos give the perfect control for this.

02 Connect the servos

The servos are still a motor, so it is advisable to give them their own power separate from the Raspberry Pi. Take note of the voltage required; most allow up to 6 volts, some less. It can share the same batteries as the motors.

03 Don't forget the kernel

To get full control over the servos, we need servod (ServoBlaster) running. So download this and make it executable with `chmod +x servod` and run it with `sudo ./servod`.

What are servos?

If the robot doesn't act like it should, or the servo goes the wrong way, just swap the servo data pins around. Double-check your code and give it another try.

> "The code we have (listed just after this section) will combine the wheel motors, servos and ultrasonic"

04 Create your script

Now we can create the test script. Swipe to the next page to see the full code listing and then start typing it in – there's a fair bit, but it's a great way to get your code-writing muscle memory working!

05 And she's off...

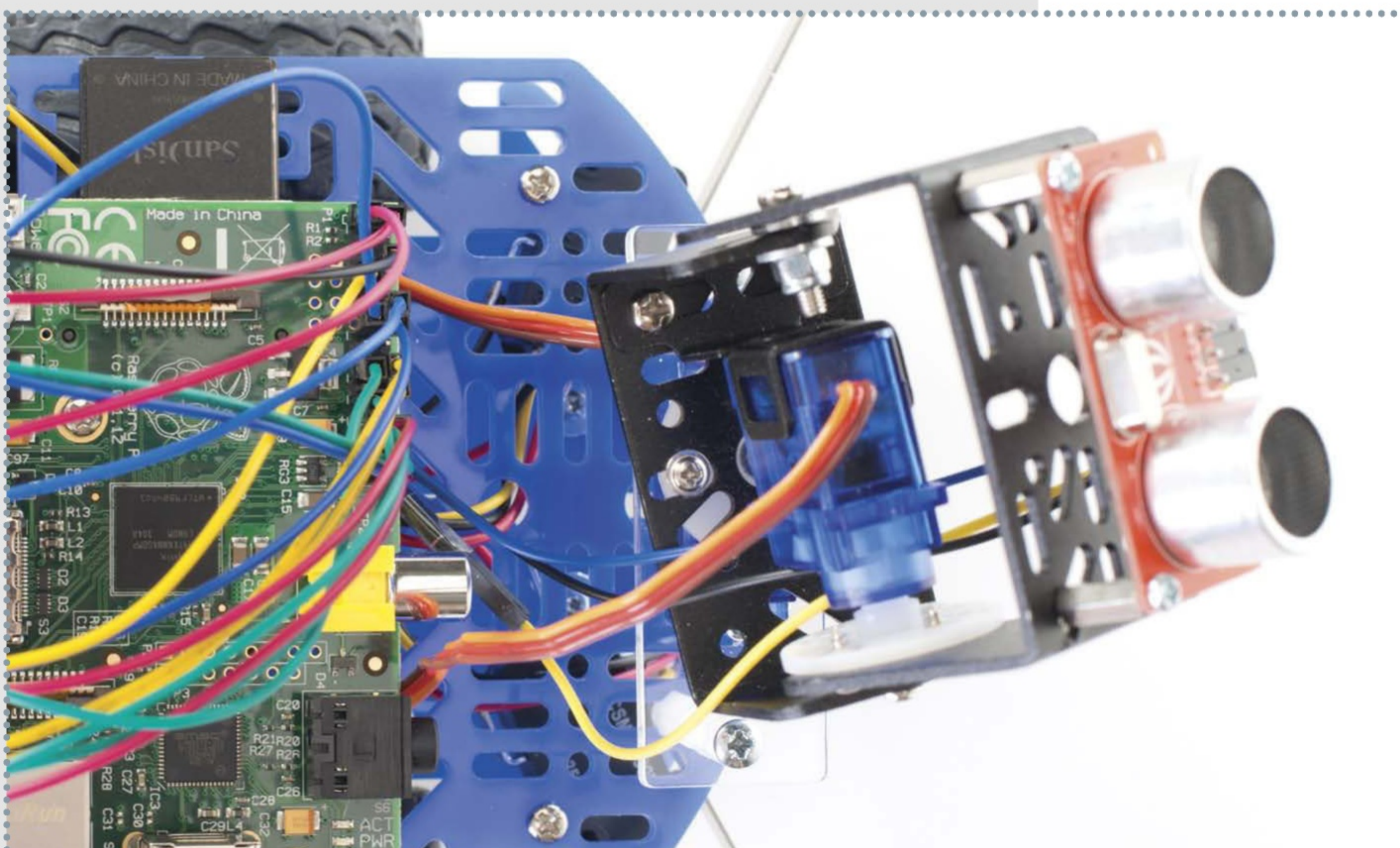
When you set off the script, the screen should fill with distance data so we can see what is happening and check on the direction it decides to take. It may pose a challenge if it gets stuck in a corner - see if you can debug it.

06 Debugging problems

If the robot doesn't act like it should, or the servo goes the wrong way, just swap the servo data pins around. Double-check your code and give it another try.

"The screen should fill with distance data so we can see what is happening and check on the direction it decides to take"

Below Test each function to ensure all motors and servos work as planned



The Code THE COMPLETE ULTRASONIC CODE

```
import RPi.GPIO as GPIO
from time import sleep
from time import time
import os

GPIO.setmode(GPIO.BCM)

GPIO.setup(24,GPIO.OUT)
GPIO.setup(23,GPIO.OUT)
GPIO.setup(25,GPIO.OUT)
GPIO.setup(9,GPIO.OUT)
GPIO.setup(10,GPIO.OUT)
GPIO.setup(11,GPIO.OUT)

Motor1 = GPIO.PWM(25, 50)
Motor1.start(0)
Motor2 = GPIO.PWM(11, 50)
Motor2.start(0)

Echo = 17
Pan = 22
Tilt = 4

def forward(speed):
  GPIO.output(24,GPIO.HIGH)
  GPIO.output(23,GPIO.LOW)
  GPIO.output(9,GPIO.HIGH)
  GPIO.output(10,GPIO.LOW)
  Motor1.ChangeDutyCycle(speed)
  Motor2.ChangeDutyCycle(speed)

def backward(speed):
  GPIO.output(24,GPIO.LOW)
```

"As we only require one GPIO pin, we will first need to set it as an output and send a 10ms pulse to trigger the sensor to start and begin counting"

The Code THE COMPLETE ULTRASONIC CODE

```
  GPIO.output(23,GPIO.HIGH)
  GPIO.output(9,GPIO.LOW)
  GPIO.output(10,GPIO.HIGH)
  Motor1.ChangeDutyCycle(speed)
  Motor2.ChangeDutyCycle(speed)

def left(speed):
  GPIO.output(24,GPIO.HIGH)
  GPIO.output(23,GPIO.LOW)
  Motor1.ChangeDutyCycle(speed)

def right(speed):
  GPIO.output(9,GPIO.HIGH)
  GPIO.output(10,GPIO.LOW)
  Motor2.ChangeDutyCycle(speed)

def stop():
  Motor1.ChangeDutyCycle(0)
  Motor2.ChangeDutyCycle(0)


def get_range():
  GPIO.setup(Echo,GPIO.OUT)
  GPIO.output(Echo, 0)
  sleep(0.1)
  GPIO.output(Echo,1)
  sleep(0.00001)
  GPIO.output(Echo,0)

  GPIO.setup(Echo,GPIO.IN)
  while GPIO.input(Echo) == 0:
    pass
  start = time()
```

"Next we switch to an input to wait for the pin to go high, at which point we stop timing and calculate how long that took"

The Code THE COMPLETE ULTRASONIC CODE

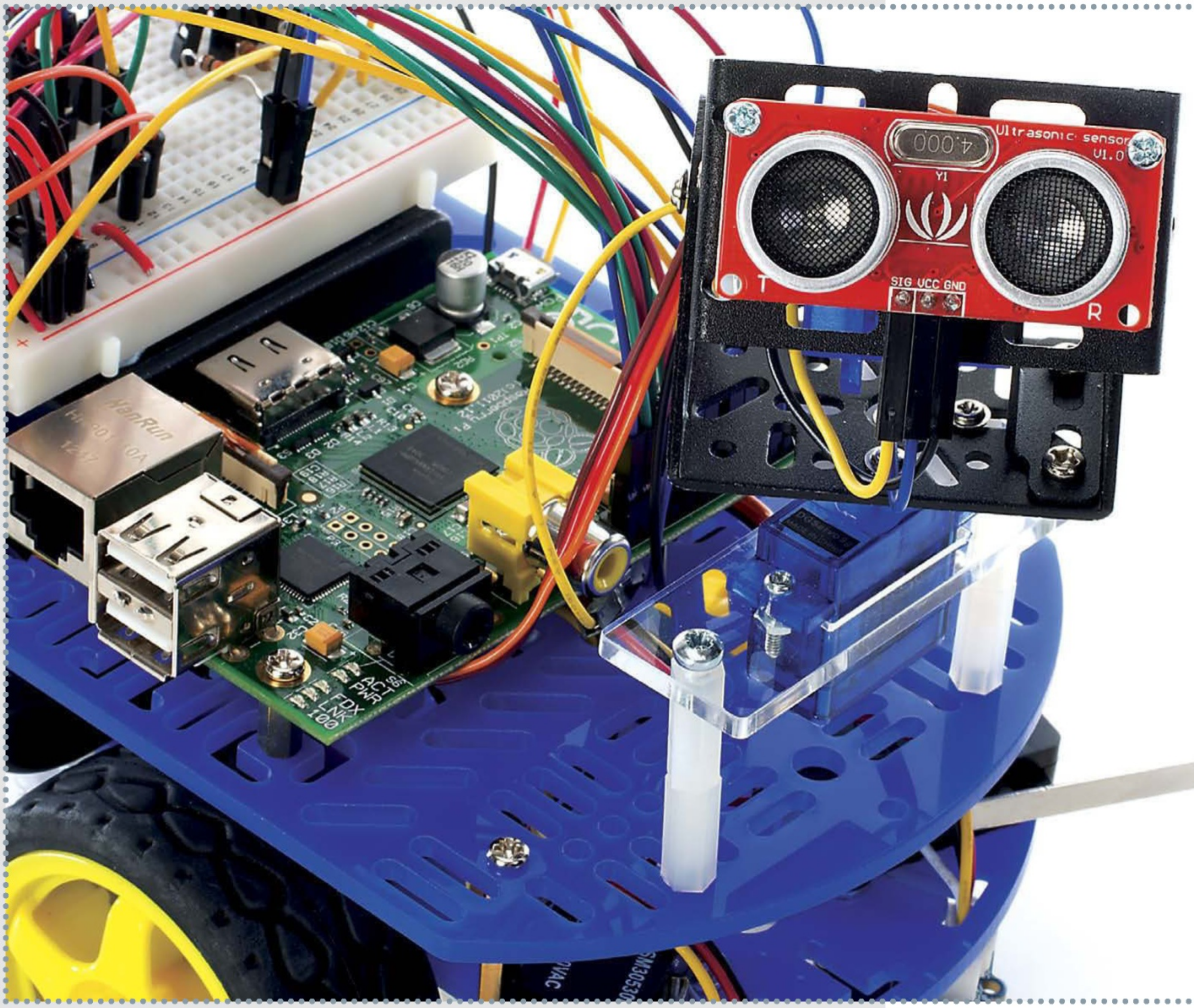
```
  while GPIO.input(Echo) == 1:
    pass
  stop = time()
  elapsed = stop - start
  distance = elapsed * 17000
  return distance


while True:
  distance = get_range()
  if distance < 30:
    print “Distance %.1f ” % distance
    stop()
    string = “echo 0=10 > /dev/servoblaster”
    os.system(string)
    sleep(1)
    disleft = get_range()
    print “Left %.1f ” % disleft

    string = “echo 0=360 > /dev/servoblaster”
    os.system(string)
    sleep(1)
    disright = get_range()
    print “Right %.1f ” % disright

    if disleft < disright:
      print “Turn right”
      left(100)
      sleep(2)
    else:
      print “Turn left”
      right(100)
      sleep(2)
```

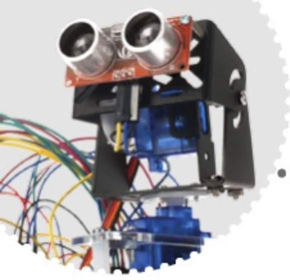
“ServoBlaster handles the timing required perfectly. All that is required is to send the angle you need to /dev/servoblaster”

The Code THE COMPLETE ULTRASONIC CODE

```
        os.system("echo 0=160 > /dev/servoblaster")

    else:
        forward(80)
        print "Distance %.1f " % distance

    sleep(0.5)

GPIO.cleanup()
```

> "If the robot doesn't act like it should, or the servo goes the wrong way, just swap the servo data pins around"





Use analogue sensors

Open your robot up to a new world of input

As we've already shown using microswitches and ultrasonic sensors, the Raspberry Pi is very capable of taking inputs and performing actions based on the outside world. Inputs also come in a variety of different types. Most common are digital sensors such as buttons and switches, but there are also analogue sensors which can be used to read temperatures or brightness. These sensors give their data in the form of a voltage.

The Raspberry Pi is unable to read an analogue signal natively, so a little help is required and this comes in the form of a microchip called an MCP3008. This chip is commonly referred to as an ADC (analogue-to-digital converter). It can communicate with the Raspberry Pi via serial and is capable of reading eight analogue inputs at once and giving their voltage in the form of a number: 0 will correspond to the lowest, while 1023 is the maximum voltage.

Using analogue, we can build a robot that is capable of following (or avoiding) bright light – perfect if you wish to have a plant pot follow the sun during the day.

"The Raspberry Pi is very capable of taking inputs and performing actions based on the outside world"

THE PROJECT ESSENTIALS

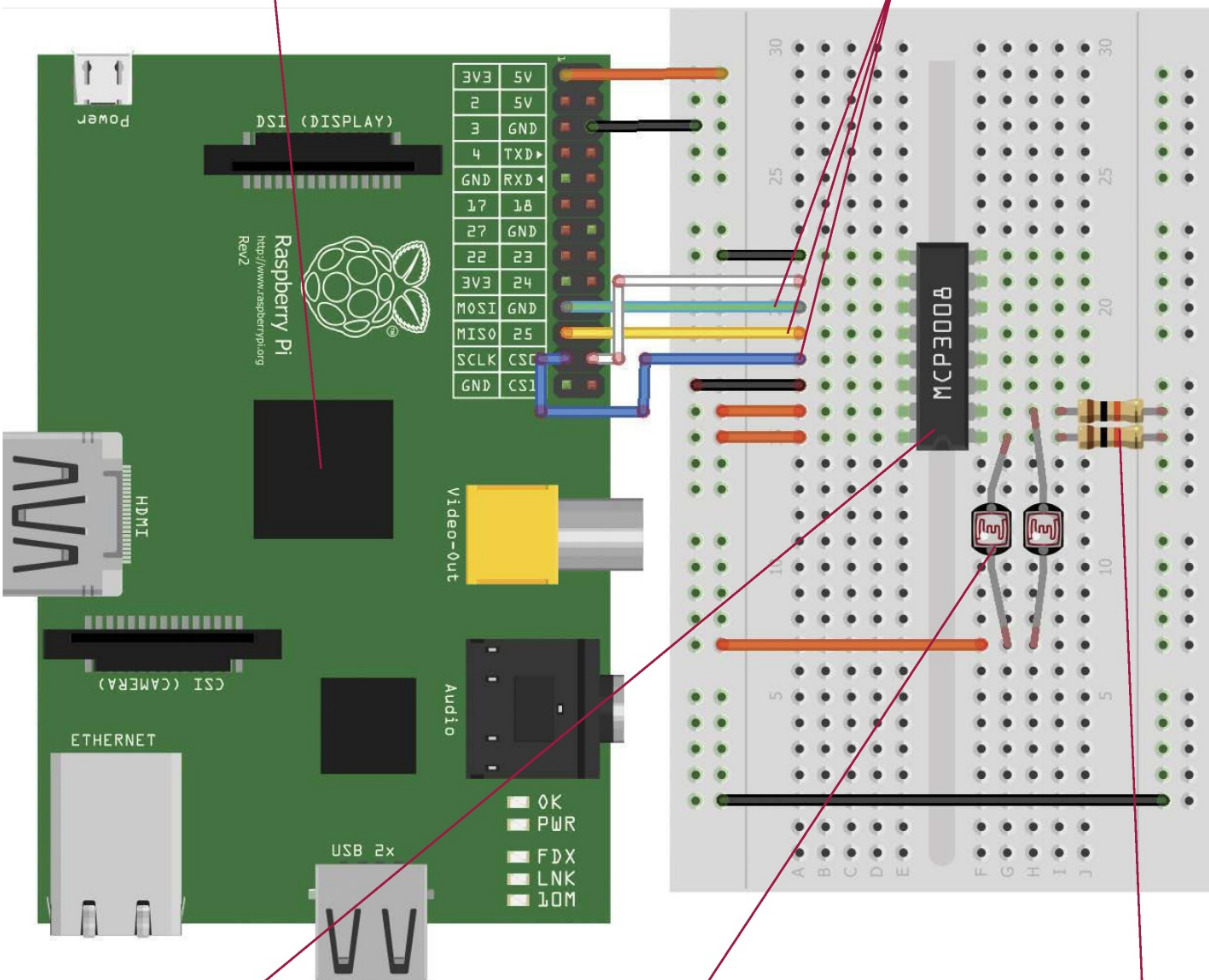
1x MCP3008
2x Light-dependent resistors (LDRs)
2x 10K resistors
Jumper wires

3.3V Power

Make sure the chip is hooked up to the 3V3 pin and not the 5V pin on the Raspberry Pi, otherwise it will kill the processor

Data cables

The MCP3008 communicates via a serial protocol called SPI, Serial Peripheral Interface. More than one can be used at the same time



MCP3008

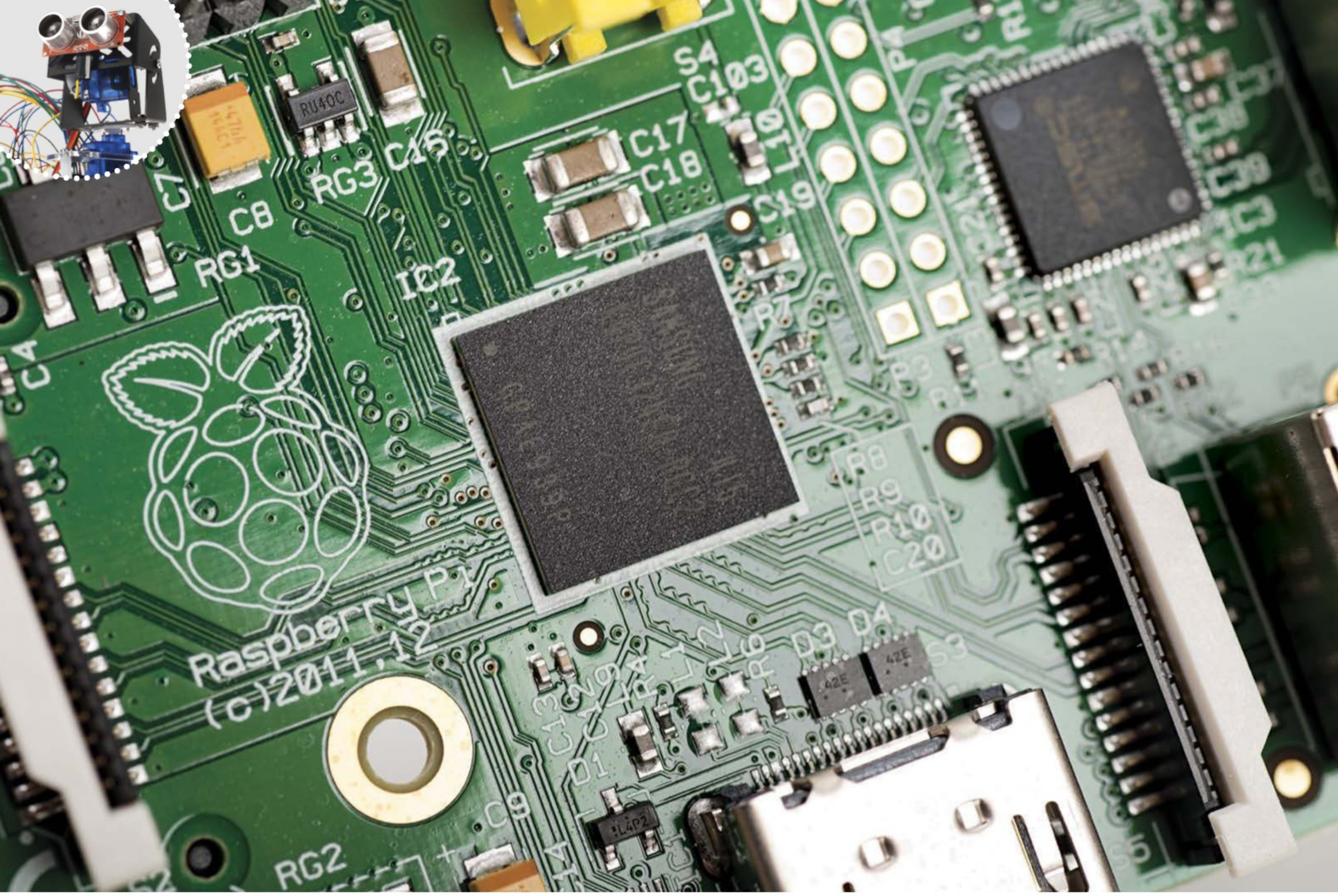
The heart of the analogue-to-digital conversion

The sensors

The light-dependent resistors (LDRs) change their voltage based on the amount of light they receive

Pull-down resistors

To give a stable reading, we have to give it a basic reference point for the voltage, so a pull-down resistor is required



Test, test and test again

Like good computer scientists we'll check it works first

Now we have wired the ADC, we need to make sure it works. So before we add it into our robot we shall use Python to read the values and display them on the screen. Doing this will help get an overall idea of what to expect when the sensor is in bright light, and how different the numbers will be when they are in the dark.

Before we can interface with the MCP3008, we need enable the serial drivers and install a Python library called spidev, so let's do this before anything else.

Open up a terminal, or connect to your Raspberry Pi, and then type in the following commands:

```
sudo nano /etc/modprobe.d/raspi-blacklist.
conf
```

And add a # to the start of each line in the file, then…

```
sudo apt-get install python-pip python-dev
sudo pip install spidev
sudo reboot
```

Once this is done, we are now free to start reading some analogue sensors!

The first two lines in our test code is to tell Python what libraries we need. Now we need to tell Python to create a new instance and tell it what channel our MCP3008 chip is on, this is handled by the next two lines.

We are nearly ready, so we'll define a function which will handle communication and returning it to our script so that we can act upon its value called 'get_value'.

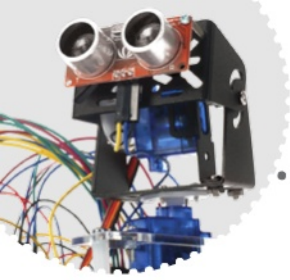
From left to right on the chip the channels start at 0 and go all the way to 7, so using this we combine with the get_value function to retrieve our value.

> "From left to right on the chip the channels start at 0 and go all the way to 7"

The Code TESTING THE ANALOGUE-TO-DIGITAL CONVERTER

```
import spidev
import time

spi = spidev.SpiDev()
spi.open(0,0)


def get_value(channel):
  if ((channel > 7) or (channel < 0)):
    return -1

  r = spi.xfer2([1,(8+channel)<<4,0])

  ret = ((r[1]&3) << 8) + (r[2] >> 2)
  return ret


while True:
  print "Chan 0: " + str(get_value(0))
  print "Chan 1: " + str(get_value(1))
  time.sleep(0.3)
```

"Make sure the chip is hooked up to the 3V3 pin and not the 5V pin on the Raspberry Pi, otherwise it will kill the processor"



Sensing light with the Raspberry Pi

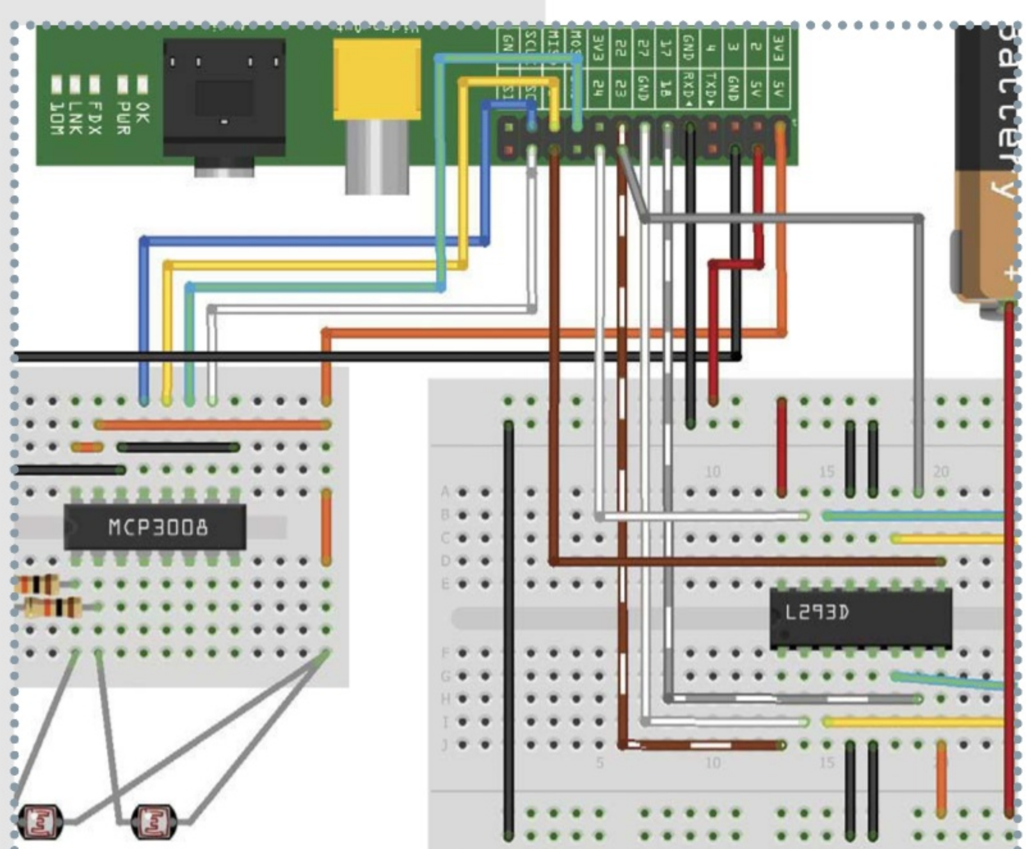
With everything connected up, let's go chase some light

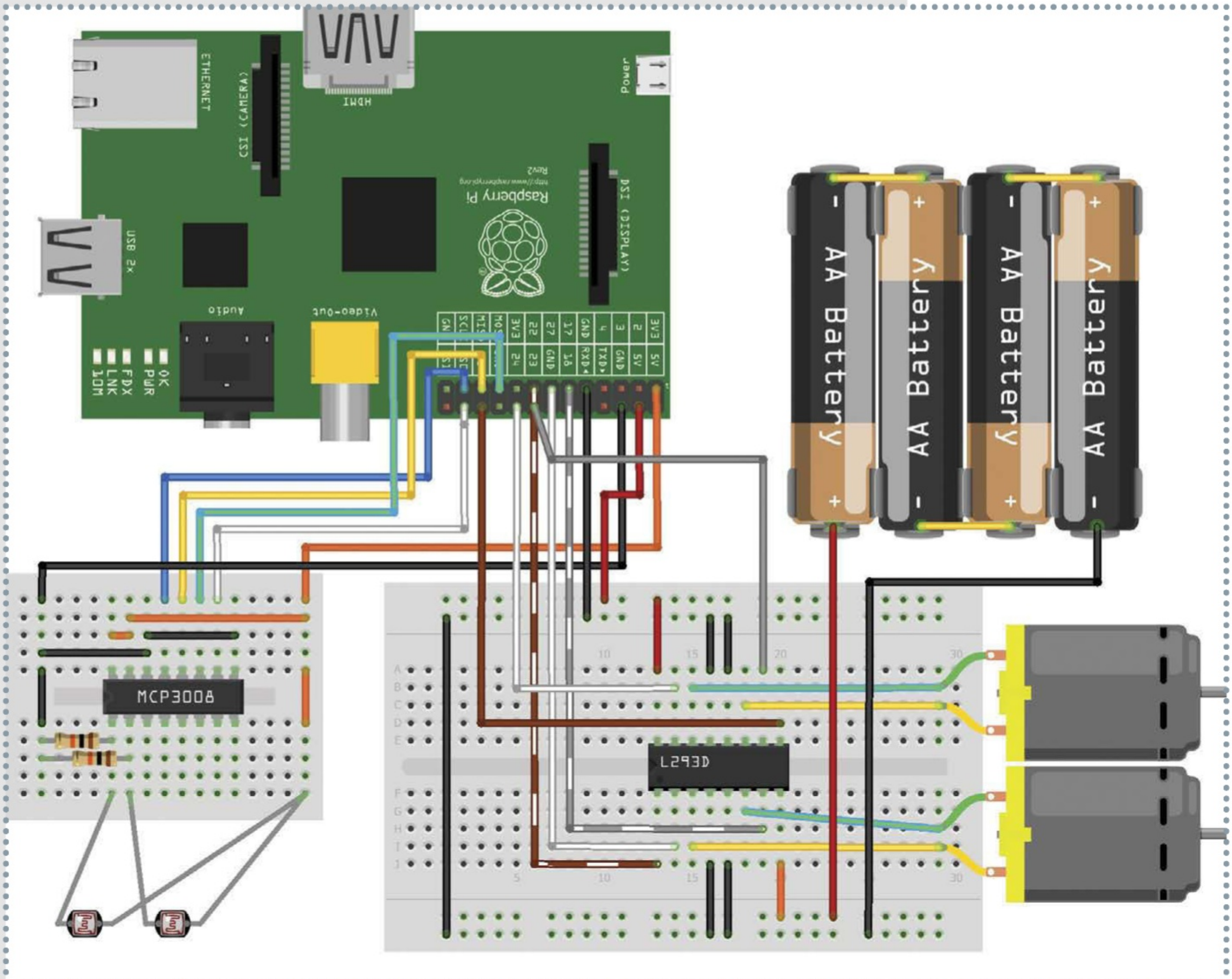
Hopefully we have a set of numbers scrolling down the screen, and have tested it by covering a sensor or shining a torch to see how it affects the readings. Now we can mount the LDRs to the front of the robot to allow it to sense the level of light. The aim now is to tell the robot to move forward at a slower pace, using a speed of 75. As the LDRs are constantly checking the light levels, if one should rise above 600 (by a torch shining at it, for instance) it will prompt the opposite wheel to speed up to turn towards the light. As each lighting situation will be slightly different, perform the test script to get an idea of the values that will be expected from the LDRs. These will fluctuate depending on the ambient light levels.

"It is best to place the LDRs apart, pointing outwards"

01 Mount the LDRs

It is best to place them apart, pointing outwards, to get a good idea of the different lighting available. If the wiring on the breadboard starts getting difficult, add another breadboard to separate the two ICs.





02 Change the motors

As we are using the SPI serial bus for the MCP3008 communication, we will need to move the motor driver pins to a different set of GPIO pins, so we shall switch pins 8, 9 and 10 to 22, 27/21 and 17.

03 Double-check everything

As we have a lot of power types – we are using 3V3 for the MCP3008, 5V to the L293D and also the batteries – it is best to check they are all correctly wired up. Once it looks good, add some power and log into the Pi.

Above Wiring can get confusing at times – if you need to, you can always add another breadboard

04 Create the script

Once everything is connected, we shall use nano to write our Python script: type

```
nano analog.py
```

...to create the file. Copy the code. Exit nano with Ctrl+X and then Y and Enter to save the file. It should be second nature by now!

05 Run the script

All that is required is to type

```
sudo python analog.py
```

...to run the program. The robot should start to follow the brightest light source. If not, check your code and connections or go back to the testing code to debug.

06 Something is wrong

If you're sure it's working properly, it could be that a tweak to the value may be needed. Run the test script again to get a suitable number to replace the 600 currently used. Remember – testing at different times of day may require you to change some of your variables.

```
  ret = ((r[1]&3) << 8) + (r[2] >> 2)
  return ret


while True:
  ldr_left = get_value(0)
  ldr_right = get_value(1)


  if ldr_left > 400:
    print "Turn right"
    right(100)
  elif ldr_right > 400:
    print "Turn left"
    left(100)
  else:
    forward(75)
```

Left If things aren't working the way they should, always go back and check your code and connections

The Code THE COMPLETE ANALOGUE CODE

```
import spidev
import time

spi = spidev.SpiDev()
spi.open(0,0)

GPIO.setmode(GPIO.BCM)

GPIO.setup(27,GPIO.OUT)
GPIO.setup(17,GPIO.OUT)
GPIO.setup(22,GPIO.OUT)
GPIO.setup(9,GPIO.OUT)
GPIO.setup(10,GPIO.OUT)
GPIO.setup(11,GPIO.OUT)

Motor1 = GPIO.PWM(22, 50)
Motor1.start(0)
Motor2 = GPIO.PWM(11, 50)
Motor2.start(0)


def forward(speed):
   GPIO.output(27,GPIO.HIGH)
   GPIO.output(17,GPIO.LOW)
   GPIO.output(9,GPIO.HIGH)
   GPIO.output(10,GPIO.LOW)
   Motor1.ChangeDutyCycle(speed)
   Motor2.ChangeDutyCycle(speed)

def backward(speed):
   GPIO.output(27,GPIO.LOW)
   GPIO.output(17,GPIO.HIGH)
   GPIO.output(9,GPIO.LOW)
```

Experiment

Try stopping the bot when the light goes below a value; or if a torch shines on an LDR sensor, spin around for 5 seconds.

The Code THE COMPLETE ANALOGUE CODE

```
  GPIO.output(10,GPIO.HIGH)
  Motor1.ChangeDutyCycle(speed)
  Motor2.ChangeDutyCycle(speed)

def left(speed):
  GPIO.output(27,GPIO.HIGH)
  GPIO.output(17,GPIO.LOW)
  Motor1.ChangeDutyCycle(speed)

def right(speed):
  GPIO.output(9,GPIO.HIGH)
  GPIO.output(10,GPIO.LOW)
  Motor2.ChangeDutyCycle(speed)

def stop():
  Motor1.ChangeDutyCycle(0)
  Motor2.ChangeDutyCycle(0)


def get_value(channel):
  if ((channel > 7) or (channel < 0)):
    return -1

  r = spi.xfer2([1,(8+channel)<<4,0])

  ret = ((r[1]&3) << 8) + (r[2] >> 2)
  return ret


while True:
  ldr_left = get_value(0)
  ldr_right = get_value(1)
```

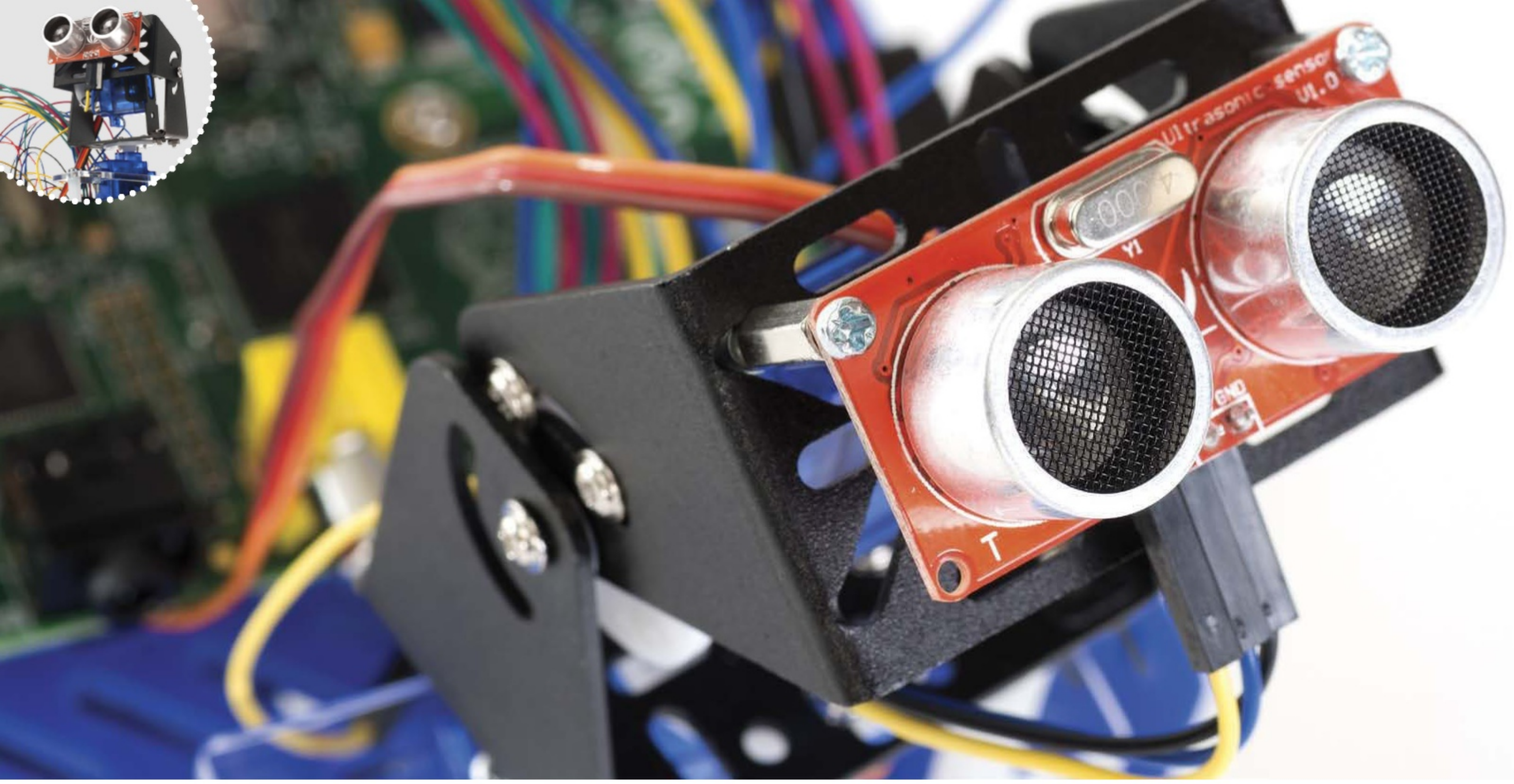
MCP3008/ MCP3004

A smaller ADC chip called the MCP3004 is also available, it only has 4 analogue channels as opposed to 8 with the MCP3008.

The Code THE COMPLETE ANALOGUE CODE

```
  if ldr_left > 600:
    print "Turn right"
    right(100)
  elif ldr_right > 600:
    print "Turn left"
    left(100)
  else:
    forward(75)

  time.sleep(0.25)
```

"As each lighting situation will be slightly different, perform the test script to get an idea of the values that will be expected from the LDRs"

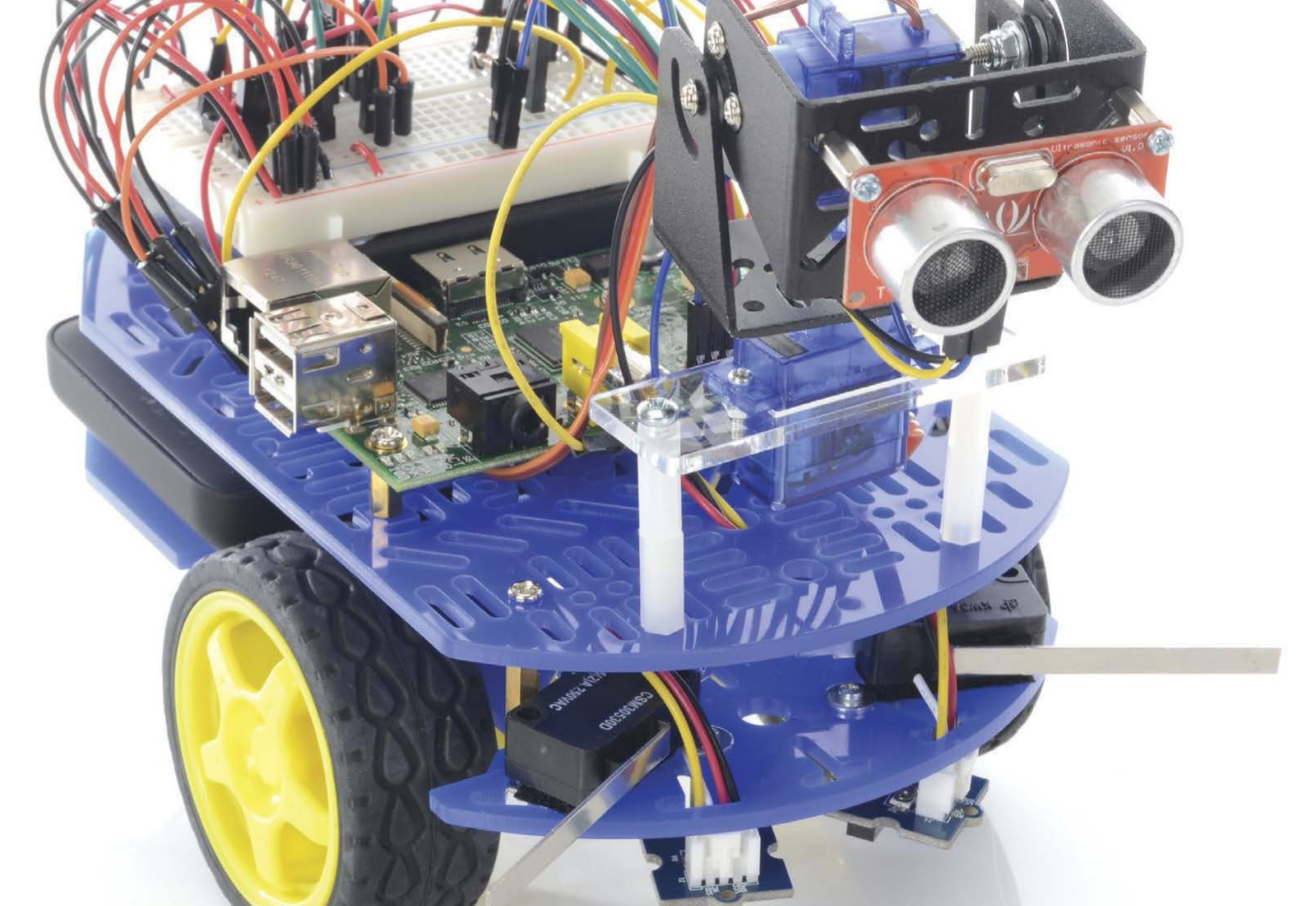


What next?

So you've finished building our project robot and you're wondering what's next...

There are loads of choices, which is one of the attractive things about robotics, and really you're only limited by your time and imagination. You could choose to expand your robot's hardware, adding more sensors as your knowledge and confidence improves, so that your robot can learn more about the world. Gas, light and sound… for practically any stimulus you can imagine, there's the corresponding sensor that you can add to your robot. With a bigger platform, you could also add an arm to your robot so it doesn't just sense the world – it can also pick up bits of the world and move them around. Here are a couple of ideas you could try…

> "With a bigger platform, you could also add an arm to your robot so it doesn't just sense the world – it can also pick up bits of the world"



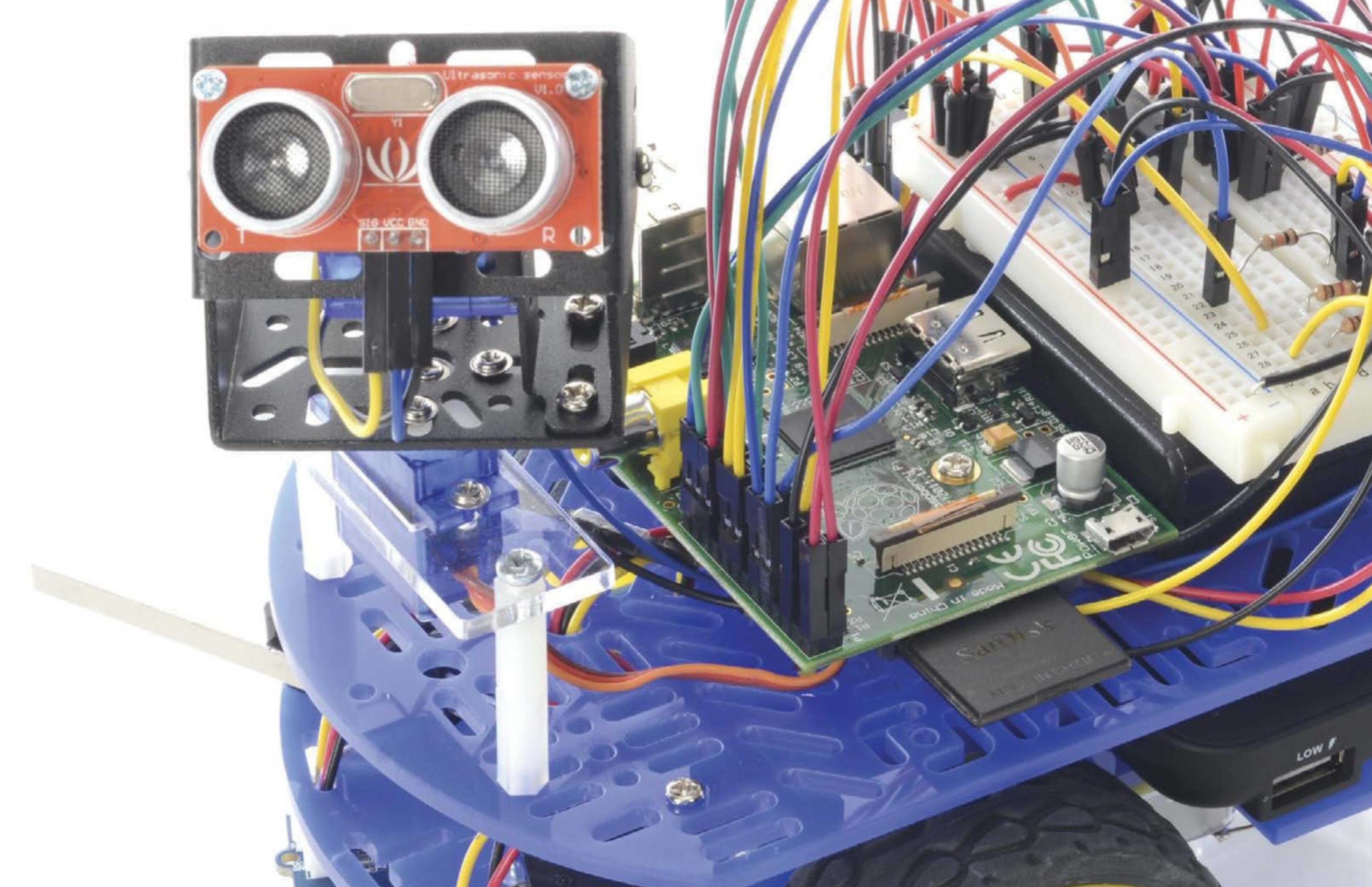
Facial recognition

With the simple addition of the Raspberry Pi's camera module and OpenCV software, face detection and recognition is possible. You could do this by replacing the ultrasonic from the pan-and-tilt mount with the camera; this will then allow the camera to move about and follow your movements.

Learning to talk

The Raspberry Pi comes with an audio output. So combining this with a travel speaker will unlock a new world of communication for your robot. Using a Python-friendly speech module like eSpeak, you can teach your robot to talk, sing or simply report readings for debugging

"The Raspberry Pi comes with an audio output. So combining this with a travel speaker will unlock a new world of communication"



purposes. This can add another human element to your creation, but adding speech recognition with a USB microphone, or similar, can take it to a whole new level.

Spatial analysis

Using the ultrasonic sensor with the pan-and-tilt kit on your robot, you can effectively measure every wall and every object in a room – a popular specialism in computer science. So by taking a series of measurements in different directions, controlled by the servos in the pan-and-tilt mount, it is possible to make a map. With another sprinkling of code and gadgetry, you could teach your bot to navigate your house. PID is an excellent field that can certainly help with this.

"Adding speech recognition with a USB microphone, or similar, can take it to a whole new level"

Swarming

Swarming is an interesting branch of computer science. Using just a little more code than we already have, we can create behaviour similar to that of a swarm of bees, or ants. A swarm of robots could discover an area quickly, or be used to scientifically model traffic-calming measures. You could even create your own synchronised routines, or build a robot football team.

Robot arm

Everyone would love a robotic helper, perfect for performing tasks around the house. Unfortunately we aren't there yet, but we can come close. By mounting a small gripper arm to the front of the robot, it can fetch lightweight items. With the help of the Pi camera module, or an RGB colour sensor, you could colour-sort LEGO bricks or entertain a pet.

Maze solving

Path finding and maze solving are other exciting branches of computer science you can attempt with your RasPi robot. Competitions are held around the world to be the fastest to solve a maze, using lines on the floor or ultrasonic sensors. All you need is a mechanism to recall past movements and a scientific approach to the robot's attitude to maze solving.

"Using just a little more code than we already have, we can create behaviour similar to that of a swarm of bees, or ants"

Bioscope

A hand-cranked film player from the 19th Century, updated for the modern era. How and why was the Bioscope created?

Where did the idea for the Bioscope initially come from?

Jon Stam: I was on vacation in Italy. I went to a flea market and I came across an old Mupi, which was the name of the toy company that produced this 8mm movie viewer. In the States, most people know it as a Fisher Price toy that already existed, but I actually never ever saw it before in my life. I knew the View-Master but never this movie viewer. I was watching an animated movie, thinking how it was amazing to see this, and then wondered about putting your own content on the device. What if your own home movies could be played in this way, where you can actually stop at the most beautiful part and speed up through the more boring parts. As soon as I came back, I went to Simon's and we agreed to do it, thinking it would be even more exciting than the View-Master.

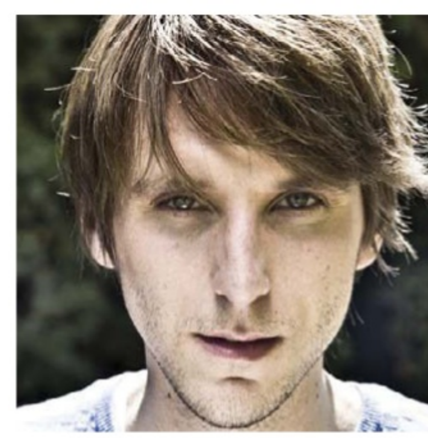
How did you two meet and create your own company?

Simon de Bakker: I met him when he was looking for technical help for one of his projects, the Imaginary Museum. It's a digitised View-Master, basically, and sort of a predecessor for the Bioscope. I did the technical part for that project. It worked quite well, actually, and we decided to continue on that basis.

JS: I started working with Simon on a few projects, and after a couple of them finished we actually realised that we work perfectly together. We have complementary skill sets and we got along, becoming really good friends. We

"The Raspberry Pi ended up just fitting exactly in the case we wanted. And it had a composite output, which we also needed"

Simon de Bakker has a background in interactive design and has been working for ten years at the V2 Institute in Rotterdam, at the V2 Lab.

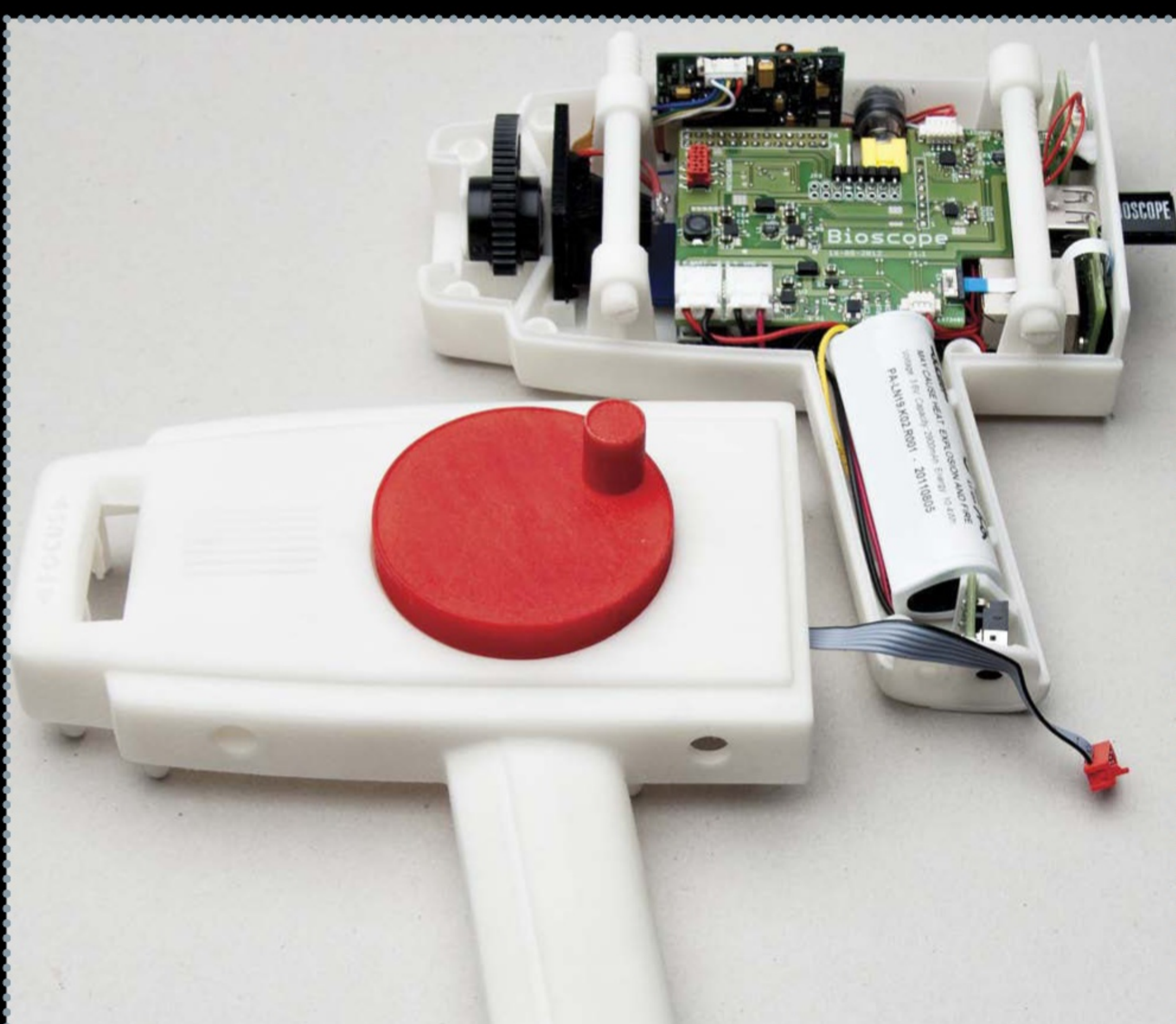


Jon Stam has studied both industrial and product design in Canada and the Netherlands respectively. In the Netherlands his design work was more artistically focused

realised if we teamed up then we could start working on more ambitious projects.

What influenced your decision to use the Raspberry Pi in the project?

SdB: When we started the project, the Pi wasn't out yet. It was just before it really came out, so it had been announced at that point. I actually had two on pre-order and we were just waiting for them to come in. We had the idea for the Bioscope and we applied for residency at the V2 Institute where I was working at that time. And we got accepted to do this project, where it was made during the summer sessions. We were looking for some hardware to actually build it – most of the boards are pretty square. The Raspberry Pi ended up just fitting exactly in the case we wanted. And it had a composite output, which we also needed. In the end, we started the project two weeks before it was supposed to be delivered. Good timing!



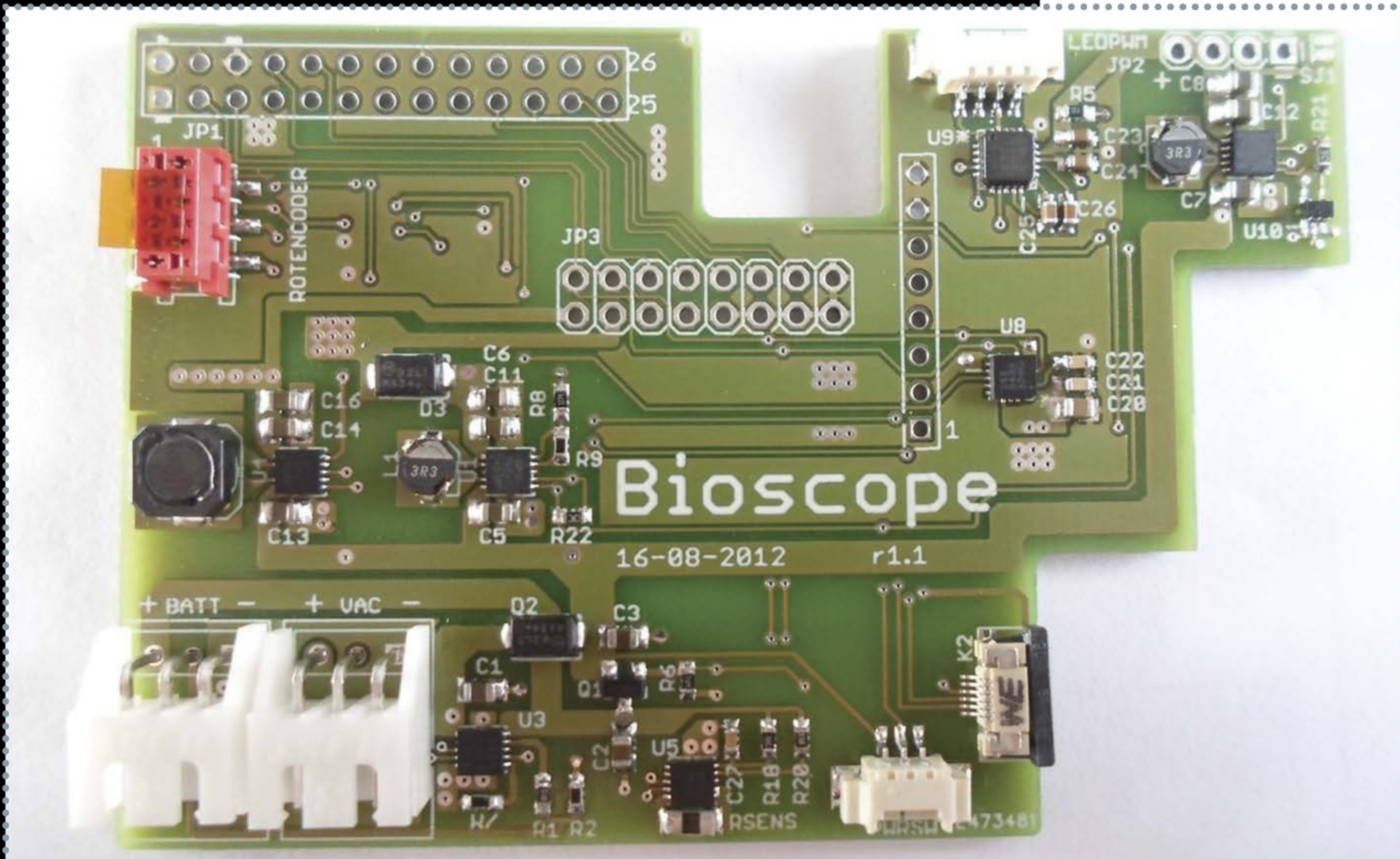
If you like

One of the side-projects resulting from the Bioscope is the Raspberry Pi Power Shield, an extremely portable and rechargeable power supply for the Raspberry Pi.

Further reading

To learn more about the Bioscope and the Power Shield, visit **www.simbits.nl**

Left The Raspberry Pi fits perfectly within the Bioscope's case, which is one of the reasons they decided to use it



Above The Power Shield is an essential part of the Bioscope, and a side project in its own right

How has it been working with the Raspberry Pi?

SdB: We're not even pushing it that much. It only displays small video in a custom format, so we can also go backwards and forwards. We're not demanding a lot from it. The biggest problem we had from it is that it's only USB powered. And we wanted to have a mobile device, a battery-powered device. What I did was design a shield with a Lithium-ion charger, a few SMPSs (switch mode power supplies) and some extras. So we could finally power it from one Lithium-ion cell, and also recharge and monitor the battery and things like that.

> "We wanted to have a battery-powered device. What I did was design a shield with a Lithium-ion charger, a few SMPSs and some extras"

Have you considered releasing it as a product?

SdB: We've been thinking about that, and actually we're pretty still a bit indecisive on that part. It's very difficult to get it into production; it looks like a simple thing, but there are a lot of different components in there.

JS: We feel that there is a potential to actually produce it in

larger quantities. However, the way that we're producing it now ends up with it being a rather expensive device. We're still actually curious and open to the different possibilities, and we've been talking to some people that actually want to commercialise it. We're not 100 per cent sure exactly where that's going or not.

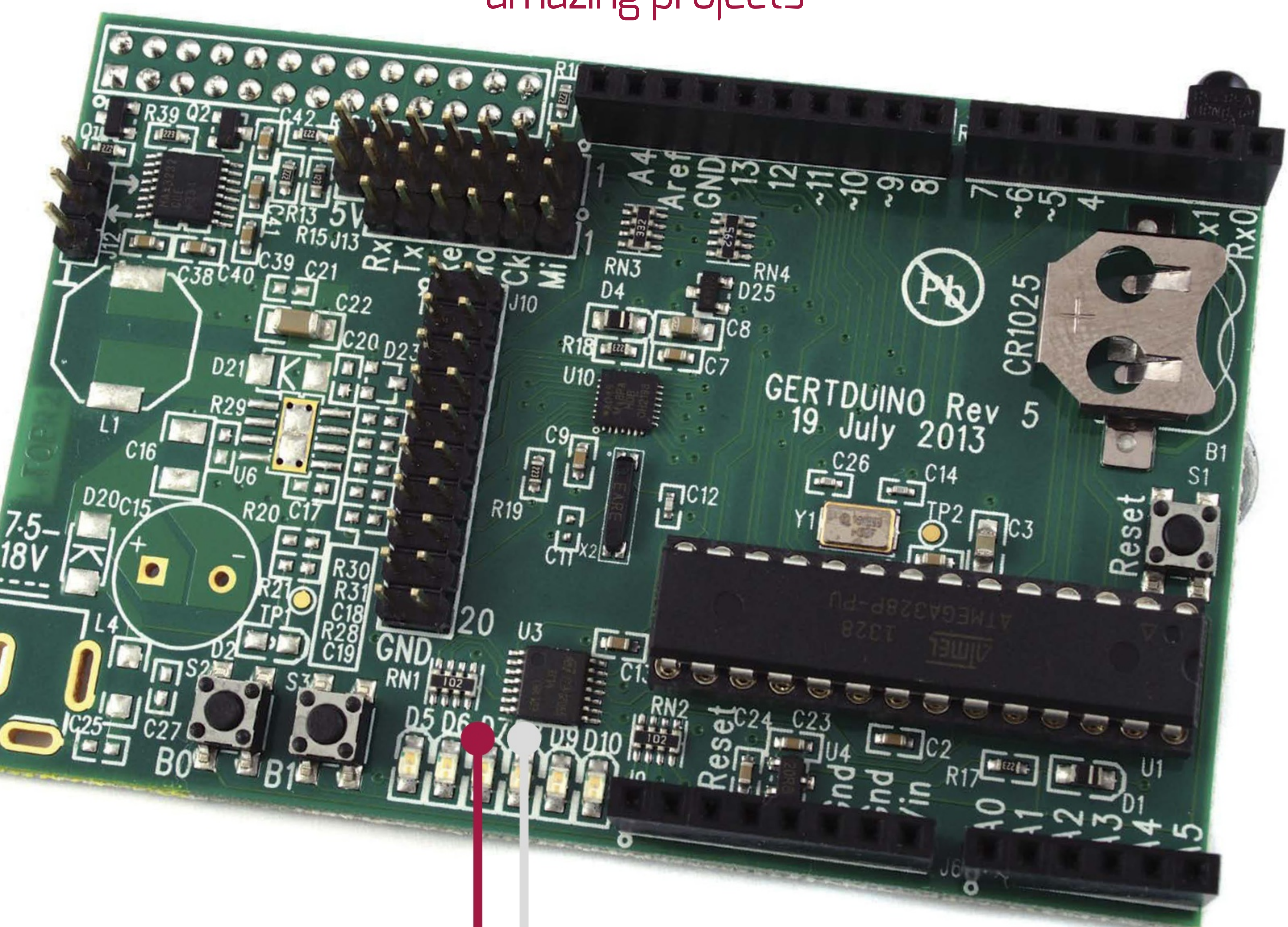
So then, what are your future plans for the Bioscope?

JS: Right now, we've been getting requests to get it into exhibitions, and also some companies are interested in just using unique ways to show their content. We're still trying to figure out where this project is going. We each have one on our shelves, but we don't want it to be a shelf project. Still, the reality is we're making actual, fully finished working products. They're not prototypes. But the whole market for the Bioscope is still in question. The bigger installations that we're doing with other projects seem to be a bit easier to get off the ground because they're is budget for them in a specific place. As it stands, we do have a few projects lined up for the near future, and right now we're working on some new stuff too. We're still interested in letting everyone be able to actually have access to the things that we're making.

"We do have a few projects lined up for the near future, and right now we're working on some new stuff too"

What is a GertDuino?

A Raspberry Pi accessory with an odd name, it's essential for making some amazing projects



"The GertDuino is capable of all of the functions that similar Arduinos can perform"

Q The GertDuino is an accessory. That's a bit of a vague term. Can you be a little more specific?
A The GertDuino is basically another circuit board that you can attach to the Raspberry Pi, extending its functionality beyond the original chips and ports and such.

Q A circuit board connected to another circuit board. I see. What's with the name, though? Sounds familiar.
A Well it's named after two things. Firstly, Gert is the name of the guy who invented it – Gert van Loo. He's a hardware engineer who's been working on the Raspberry Pi since the alpha boards, and currently still volunteers at the Foundation. He's an active mod on the forums, and worked on the Raspberry Pi camera board as well.

Q Oh yeah, didn't he release another circuit board with his name on?
A That's correct: Gert also invented the Gertboard, another Raspberry Pi accessory.

Q So how is that different from the GertDuino? Is the GertDuino a Gertboard upgrade?
A Not exactly. The Gertboard extends the functionality of the GPIO ports on the Raspberry Pi, allowing you to power motors, detect switches, illuminate LEDs and generally enable it to interact with the physical world. The GertDuino does a bit more than that.

Q So the Gert part I now get, what's with Duino?
A Duino as in Arduino, the open source microcontroller platform. GertDuino uses an Arduino microcontroller.

Q That sort of answers my question, but what exactly is a microcontroller?
A A microcontroller is a bit like a CPU; however, it

What makes up GertDuino?

The individual components that a GertDuino replaces:

Arduino Uno
The crown jewel of Arduino microcontrollers right now, it's used by a lot of people to power a lot of Arduino-based projects. It uses the ATmega 328 chip, the same as can be found on the GertDuino.

Real-time clock
Using an ATmega 48 chip, the GertDuino can offer the extra functionality of a high-precision clock, infrared data association (IRDA, used for remote controls) and even a backup battery to help keep the memory and the clock alive.

has specific inputs and outputs that are specifically programmable for the task at hand. It's a lot more cost- and space-efficient than an entire computer system built up to do it, as it manages to do all the operations on a single chip. The type of microcontroller used in the GertDuino has a lot of physical circuits, which means that people do not need to build the physical board up around the microcontroller.

Q That sounds pretty neat. What's the deal with it being Arduino then?

A Well, as we said, it's open source. Open hardware. The specs are there for everyone to use, so it's easily implemented into the GertDuino with no licensing fee. Arduino is also extremely popular as microcontrollers go, making it easier for people to pick it up if they've got experience with it.

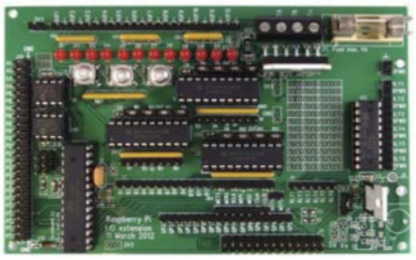
Q Experience with it? Is there anything particularly special about their operation?

A Arduinos can be controlled with the Arduino IDE, a cross-platform language that is designed to be slightly easier to use for people who don't have any formal training in coding. Even if you do write code in C, either way it allows you to upload programs, or a sketch, to the Arduino board itself.

Q Okay then, that also sounds pretty good. So the GertDuino adds an Arduino chip to the Raspberry Pi. Isn't the Raspberry Pi already programmable and small anyway?

A It's not quite that simple – yes, the Raspberry Pi is a full computer on a chip. However, it lacks the necessary functions and I/O ports to be used the same way as an Arduino chip/board. It's sort of like trying to print because

What makes up GertDuino?



Gertboard
In some ways a very basic version of the GertDuino, the Gertboard is one of the first add-ons that allowed the Raspberry Pi to do some physical computing. The GertDuino board manages to take this to another level.

you have all of the software, but you still don't have an actual printer.

Q I'll take your word for it. How does the GertDuino connect to the Raspberry Pi?

A Like the Gertboard and most other major add-ons, it's connected via the GPIO ports. These are the major expansion ports for the Raspberry Pi, and it allows the Pi to properly speak with the Arduino chip. This allows you to actually then program the Pi, which will then program the Arduino to do the work you want it to do.

Q Can you not connect a normal Arduino up to a Raspberry Pi then?

A Actually, you can. People have been using the Raspberry Pi to help power Arduino boards since it first came out.

Q So what's the upside of using a GertDuino over a standard Arduino board?

A Well for one, it's actually pretty cheap compared to an equivalent Arduino board, although it's tied a bit to the Raspberry Pi as a result. Secondly, it offers the same functionality as one of the major Arduino boards, the Arduino Uno. It's also completely compatible with everything that works with the Uno, and offers a few more features on top of all that.

Q A few more features? Sounds promising. What would those be then?

A Well, it has a special chip on it called an ATmega 48, which offers infrared signal support, a very precise real-time clock as well as a form of spare battery. The main chip is called the ATmega 328, by the way. It can also be unplugged from the Raspberry Pi once you're done

"It has a special chip on it called an ATmega 48, which offers infrared signal support, a very precise real-time clock as well as a form of spare battery"

programming it, and it will work on its own thanks to the ATmega 48's battery.

Q So it sounds pretty nifty, but what can I use it to do?

A Arduino powers a lot of physical hobby projects. Garage door openers, robots, home breweries and even some of the Makerbots. The GertDuino is capable of all of the functions that similar Arduinos can perform. Gert van Loo himself has suggested that you can use it to time the opening of a chicken coop in the morning, or as a remote for opening and closing curtains.

Q Just for hobby projects? What about full commercial projects?

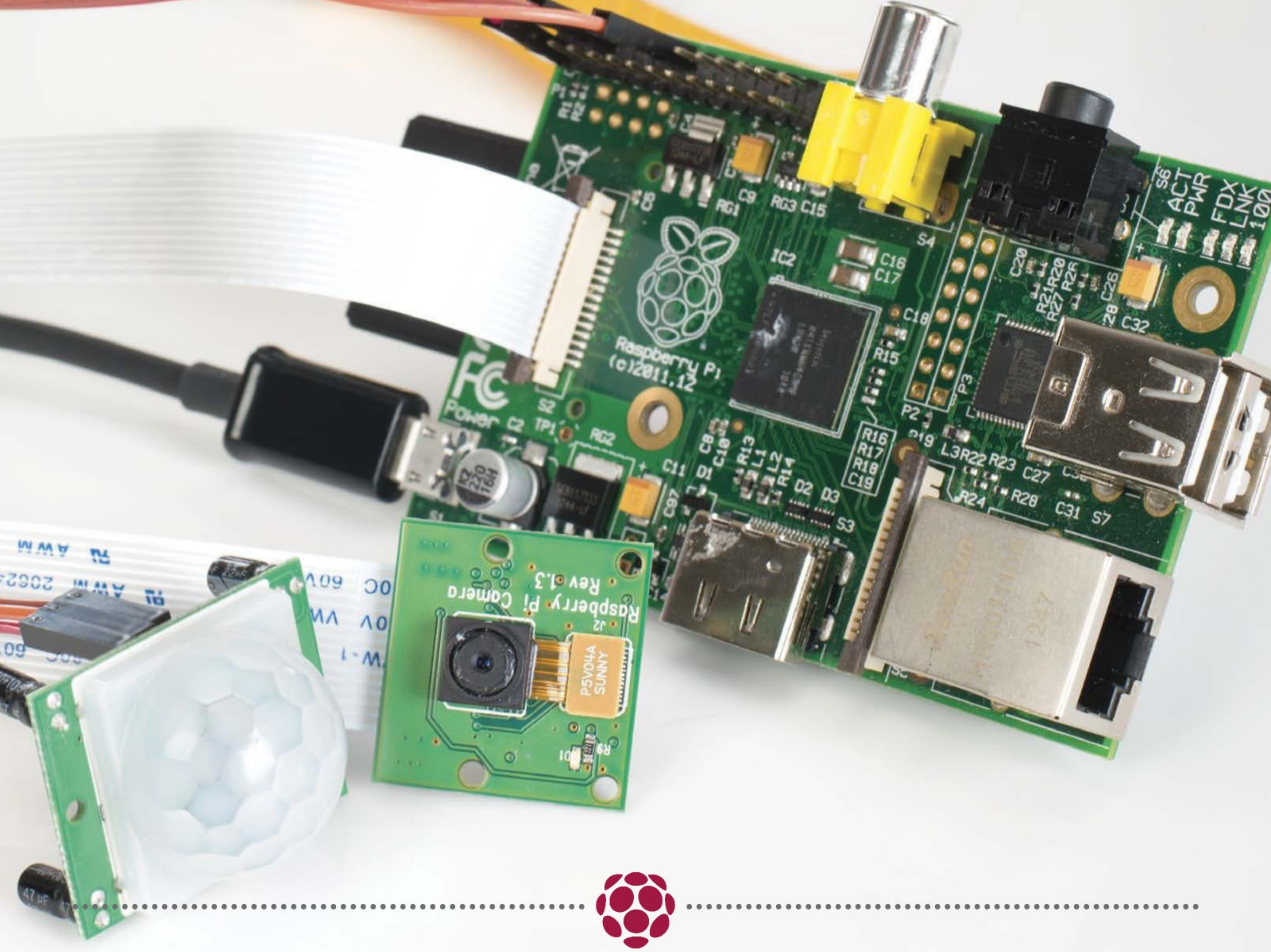
A While Arduino boards are fantastic for rapid-prototyping and tinkering with low financial risk, when your product is complete it is usually a lot cheaper to start making custom chips. For everyone else, though, it's the most cost-effective for both prototypes and final products.

Q All right then, I think it's time to get one. My Rube Goldberg alarm clock needs upgrading. Where do I get one of boards?

A You can buy a board from element14, which coincidentally is one of the four major distributors of Raspberry Pis. The manual for the board is also available there, along with some basic guides on how to use it, and more useful information on the board itself.

> "Arduino powers a lot of physical hobby projects. Garage door openers, robots, home breweries and even some of the Makerbots"

Below The GertDuino is designed to sit on top of the Raspberry Pi, so you'll be able to fit both into a case if you want to



The tweeting motion sensor

Create an Internet of Things device able to takes pictures of wildlife before tweeting them to the web

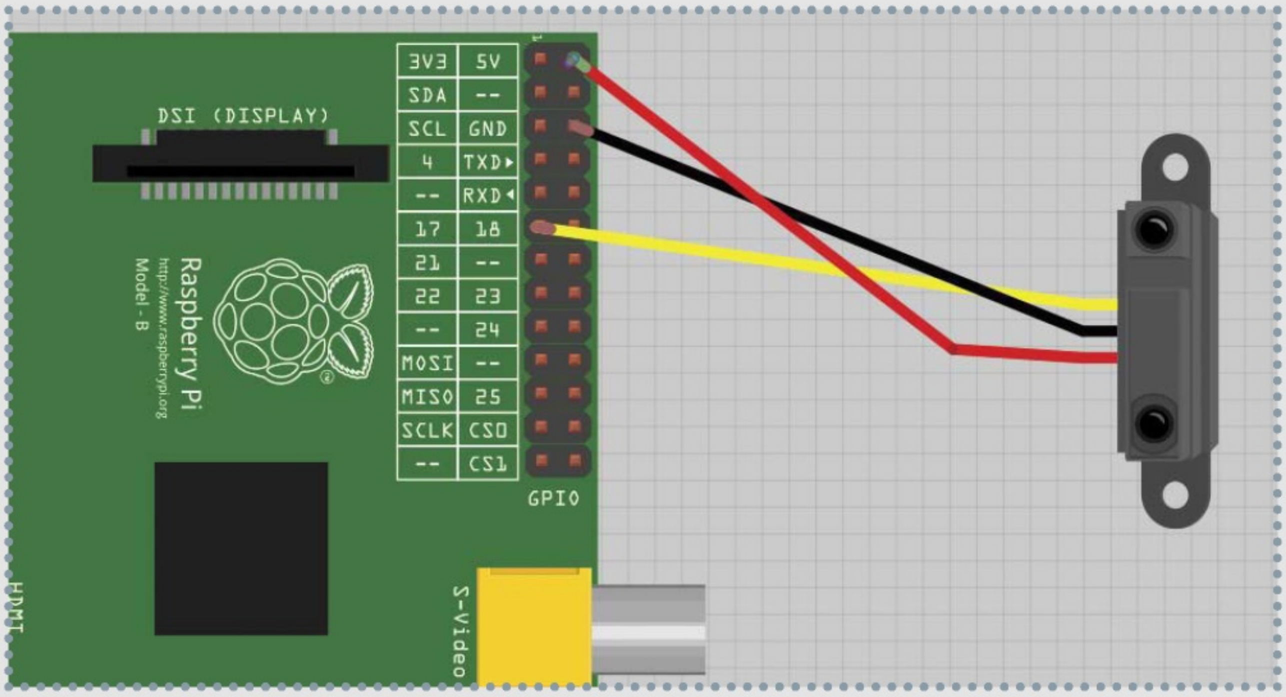
This project uses lots of the skills and technology we've covered in RasPi so far and raises the bar to include things like automated Twitter updates and event detection. We're going to create our own little Internet of Things device that incorporates a simple PIR sensor (just £2.99/$5 from **modmypi.com**), the Raspberry Pi Camera board and the power of the internet

THE PROJECT ESSENTIALS

tweepy
Internet connection
HC-SR501 PIR Infrared sensor
Camera board & picamera library

to let us automagically capture images of birds (and other wildlife) before tweeting to a Twitter account of our choice whenever activity occurs.

01 Configure the infrared sensor



Since this an automated device, we need a way to trigger the camera when movement is sensed in our camera's target area. One affordable and easy-to-configure solution is the HC-SR501 Infrared Motion Sensor. The device itself has three pins – VCC (5V), Ground and a signal pin that sits in the middle. We've configured our script for the pir pin to trigger GPIO pin 17 (it sits opposite the PWM pin). The VCC pin is connected directly to the 5V power pin and the Ground to the same Ground pin we've used throughout the tutorials.

02 Test the PIR

We don't want to send millions of accidental tweets to Twitter, so we should do some pretty extensive testing with the PIR first, using simple print statements to show when motion has been detected. Visit **github.com/linuxusermag/tweety-pi** and find the `pir_`

"We don't want to send millions of accidental tweets to Twitter, so we should do some pretty extensive testing"

`testing.py` script. Copy it onto your Pi and run it with your PIR connected. You'll probably find that it's over-sensitive for your needs by default. You'll find two tiny adjustable screws on the PIR. Gently adjust them to the left to lower the sensitivity and test thoroughly until you get the desired result.

"You'll find two tiny adjustable screws on the PIR. Gently adjust them to the left to lower the sensitivity"

03 Set up the project

With the camera connected as per our previous camera project, we need to tie the camera, motion sensor and Twitter code together to make a tangible project that can be left to do its job. We'll walk through the script, but it's worth looking around our project by cloning it from GitHub. In the terminal type:

```
git clone https://github.com/linuxusermag/
tweety-pi.git
```

Enter the project with `cd tweety-pi` to take a look around. It's been set up as a full (if a little basic) project with a readme, licence and even a folder for our pictures to sit in.

04 The callback function

One of the big changes in this project compared to our previous ones is that we're using the GPIO as an input, instead of an output. Since we want the PIR sensor to alert us to movement, we really want the PIR to interrupt our script to let us know – that's where our callback function `motion_sense()` comes in. Looking further down the script to the main program loop you'll see a `GPIO.add_event_detect.` Whenever the assigned GPIO pin gets pinged, the script will stop what it is doing and jump to the named callback function (in this case `motion_sense`). This simple function then calls the `take_picture` function below it.

05 Simple chain

The entire chain of main functions that make up the meat of the project are laid out in trigger order and all initiated from that initial callback function. Once motion is detected the `take_picture` function is called. As soon as the image has been saved to the `/pics` folder we call the `update_twitter` function. Here, we're loading our previously saved image and using the Twitter API's `update_with_media` method to allow us to tweet our picture to the outside world. We can set our status from within this line, but instead of repeating the same phrase we use the random module's choice method to pick from a list of three we'd assigned to the variable `tweet_text` earlier in the script.

05 Main program loop

As we've done before, we're placing our main program loop in `try, except, finally` blocks to ensure we can cleanly quit the program or clean up should it crash for any reason. After we call our GPIO event detect line, we create a simple infinite loop to ensure the script keeps running. Pressing `Ctrl+C` will break this loop, causing the program to end, but not before finally calling the methods that close the camera and shut-off the GPIO pins. If you don't do this, all kinds of issues can arise the next time you run the script. And that's all there is to it! Be sure to use your knowledge on experimenting with other Raspberry Pi projects – and have fun!

"After we call our GPIO event detect line, we create a simple infinite loop to ensure the script keeps running"

The Code THE TWEETING MOTION SENSOR

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import random, time, os
import tweepy
import picamera

pir = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(pir, GPIO.IN)

### TWITTER SETTINGS ###
# Set your access keys via https://apps.twitter.com
api_key = 'your_api_key_number'
api_secret = 'your_api_secret_number'
access_token = 'your_access_token_number'
token_secret = 'your_token_secret_number'
auth = tweepy.OAuthHandler(api_key, api_secret)
auth.set_access_token(access_token, token_secret)
api = tweepy.API(auth)
my_twitter = api.me()
print my_twitter.name, "is connected! Press CTRL + C to quit."
# Three  statuses. We'll pick one at random to go with our pic
tweet_text = ['Another shot taken with tweety-pi!',
            'Just spotted with my Raspberry Pi',
            'Snapped automagically with my Raspberry Pi camera!']

### CAMERA SETTINGS ###
camera = PiCamera()
cam_res = (1024, 768)
camera.led = False # Turn off LED so we don't scare the birds!
pics_taken = 0
time.sleep(1)
```

"The entire chain of main functions that make up the meat of the project are laid out in trigger order and all initiated from that initial callback function"

The Code THE TWEETING MOTION SENSOR

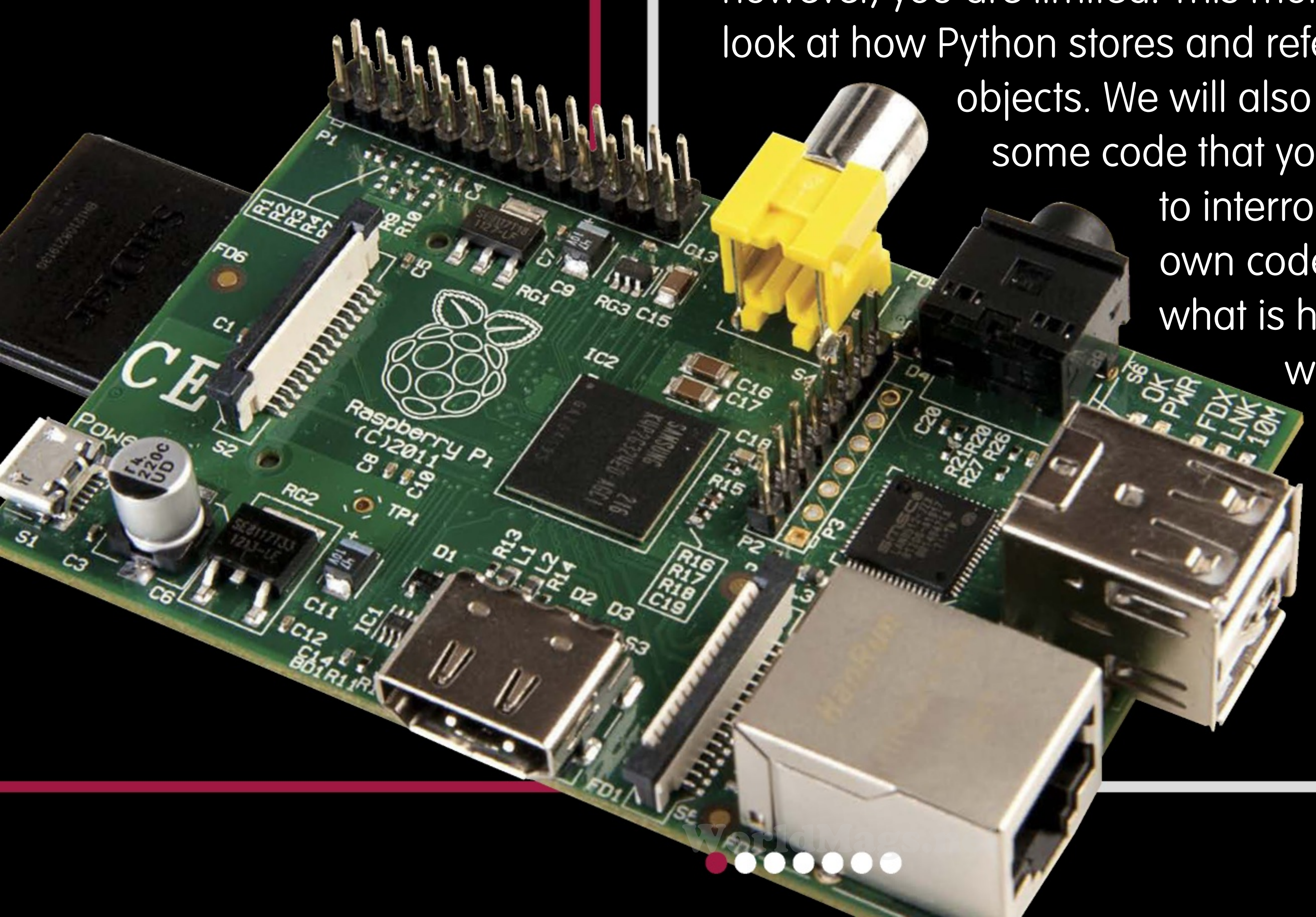
```
### MAIN FUNCTIONS ###
def motion_sense(pir):
  print "Motion detected... Taking picture!"
  take_picture(cam_res)

def take_picture(resolution):
  global pics_taken
    camera.resolution = resolution
    # Capture a sequence of frames
    camera.capture(os.path.join(
      'pics', 'image_' + str(pics_taken) + '.jpg'))
    pics_taken += 1
    print "Picture taken! Tweeting it...""""
    update_twitter()

def update_twitter():
  api.update_with_media(os.path.join(
    'pics', 'image_' + str(pics_taken -1) + '.jpg'),
    status = random.choice(tweet_text))
  print "Status updated!"
  #We don't want to tweet more than once per minute!
  time.sleep(60)

### MAIN PROGRAM LOOP ###
try:
  GPIO.add_event_detect(pir, GPIO.RISING, callback=motion_sense)
  while True:
    time.sleep(60)
except KeyboardInterrupt:
  print "\nQuitting"
finally:
  camera.close()
  GPIO.cleanup()
```

Raspberry Pi: not unlimited

Memory is a limited resource on the Raspberry Pi. What can you do to make the most of it in Python?

> "With most basic object types, the getsizeof function will give you the total amount of memory used"

In the first issue of **RasPi**, we started with a quick look at Python objects, including an introduction to creating your own. One thing that we only saw in passing was just how ubiquitous objects are in Python. Pretty much everything in Python is an object. If you have gigabytes of memory, how these objects get stored is not a major issue. On a Raspberry Pi, however, you are limited. This month, we will look at how Python stores and references objects. We will also look at some code that you can use to interrogate your own code to see what is happening with RAM usage.



The first thing to realise is that everything in Python is an object, of one type or another. You can find out the type of an object with the command `type()`. If you were to create a list of integers with the command `a=[1,2,3]`, running the command `type(a)` would return that list. But it goes even further than that. What happens if you run the command `type(1)`? Does the integer 1 have a type? In this case, you will get the result 'integer'. This makes sense. But it goes even further. The integer 1 is actually an object. Integer objects have a function named `bit_length()`, which gives the number of digits needed to represent this integer in binary. If you have `num1=1`, you can find the bit length with `num1.bit_length()`. But the interesting thing is that, since the integer 1 is an object, you can also execute `(1).bit_length()`. This is very different behaviour from most other programming languages you'll encounter.

Since everything is an object, what are variables? Variables are simply labels, pointing to the objects that they represent. This means that you can have more than one variable pointing to a particular object. If we take the list we defined above, we can create a new label pointing to it with the line `b=a`. Now both variables, a and b, point to the same object. You can prove this to yourself by adding something to the end of b. If you run `b.append(4)`, typing in `a` will give you the list `[1, 2, 3, 4]`. This is good to know. You can create new references to objects without accidentally creating extra copies and using up memory unnecessarily. But this also means that if you actually wanted to make a copy, you need to do it explicitly. For lists, you can do this with slices.

Slicing lists

Slices are used to get a subset of the contents of a list and return them in a new list. The format is `[start:end]`,

> "Since everything is an object, what are variables? Variables are simply labels, pointing to the objects that they represent"

where 'start' is the beginning index of the slice and 'end' is the finishing index of the slice. If you leave out 'start', then the implied index is 0, and if you leave out 'end', the implied index is the length of your initial list. Knowing this, you can get a copy of `a` with `b = a[:]`. Now if you alter `b`, you will not be affecting `a`. But, if it is so easy to generate references to objects, how can you keep track of how many there are? One of the modules that is included with Python is `sys`. Once you import it, you have a set of functions that allow you to interact with and query the system that the Python engine is running on. The specific one that will help us here is `getrefcount()`. If we run `sys.getrefcount(a)`, we should see a result of 2 – one for the variable `a`, and one for Python's reference to the object that a points to. If you were to add another reference with the command `c=a`, re-running `getrefcount()` would give a result of 3. It is interesting to run `sys.getrefcount(1)`. You will likely see several hundred, or even several thousand, references to the object `1`. Yet more evidence that even raw integers are actually objects.

"One of the modules that is included with Python is sys. Once you import it, you have a set of functions that allow you to interact with and query the system"

Memory matters

The other thing that is concerning to Raspberry Pi users is how much memory is being used by all of these objects. Since we already have the `sys` module imported, we can use that to check this out. Another function available is `sys.getsizeof()`. This function will return the number of bytes being used by the object in question. On your advisor's system, the size of an integer is 8 bytes. You can check this with `sys.getsizeof(1)`. With most basic object types, `getsizeof` will give you the total amount of memory used. So, for example, if you have an empty list with `a=[]`, `sys.getsizeof(a)` gives an answer of 72. Adding an entry with `b=[1]` gives a size of 80. A

list with two elements takes up 88 bytes. So, the basic size of a list with all of the required metadata is 72 bytes, and each additional integer adds 8 bytes to the size. Unfortunately, `getsizeof` doesn't work as well if the object in question is compound. If you have a list of strings rather than integers, this becomes evident very quickly. Executing `sys.getsizeof('a')` gives 38 bytes. But, if we stick this string in a list first and then run `getsizeof`, it seems to only take 8 bytes. Obviously, what `getsizeof` is measuring is actually the size of the variable pointing to the string. To get the complete size of the list, you will need to loop through all of the elements and get the size of each individually.

The other measure of RAM is how much is being used by the Python interpreter as a whole. You can get this by importing the resource module. In the sample code, you will find a function that uses this module to get the total amount of RAM being used. Unfortunately, this method gives you the maximum amount used up to this point, so you can't see what happens if you try to clean up your memory usage. In order to do this, you will need to use a different module, such as `guppy`.

Taking out the trash

When dealing with memory and object oriented programming languages, one thing that comes up is the concept of garbage collection. Whenever you have the ability to reference an object with more than one label, you need to keep track of those references. The system can't free the memory until all references have been removed. In Python, you have control over how garbage collection is done. To get this control, you will need to import the `gc` module. You can turn garbage collection on and off with the functions `gc.enable()` and `gc.disable()`. If you want to see how many objects

"Whenever you have the ability to reference an object with more than one label, you need to keep track of those references"

are being tracked by the garbage collector, you can use the function `gc.get_count()`. Normally, garbage collection is handled automatically by an algorithm that is meant to maximise memory usage with minimal impact on runtime. But, if you want to force a garbage collection, you can do so with the function `gc.collect()`. The number of objects cleaned up is returned. You can set the threshold levels used by the garbage collector with the function `gc.set_threshold()`. You can always check what objects are about to be cleaned up with the variable `gc.garbage`. With the `gc` module, you get a lot more control over your system than in most languages.

Below With projects like a Pi-powered buggy, the memory freed up by garbage collector can make all the difference

The Code MANAGING MEMORY

```
# First, you will need to import the module sys
import sys

# What is the size of an integer?
sys.getsizeof(1)

# A string?
sys.getsizeof('a')

# How about lists?
a = []
sys.getsizeof(a)
a.append(1)
sys.getsizeof(a)
a.append('abc')
sys.getsizeof(a)

# Can we count references?
b = []
sys.getrefcount(b)
c = b
sys.getrefcount(b)
# Do you get the same from the other variable?
sys.getrefcount(c)

# How much RAM are you using
import resource

def memory_usage():
    rusage_denom = 1024.
    mem = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss / rusage_denom
    return mem
```

Sizing things up

It's possible to check the size of any given variable in Python using the sys module's 'getsizeof' method. See **www.python.org/doc** for more details.

The Code MANAGING MEMORY

```
memory_usage()
list1 = range(10000)
memory_usage()
list2 = range(1000000)
memory_usage()
# Do we use up more RAM if we make another reference?
list3 = list2
memory_usage()

# Don't forget to cleanup when you are done
a = b = c = []
list1 = list2 = list3 = []

import gc
# How much garbage?
gc.garbage
# Go ahead and cleanup
gc.collect()
```

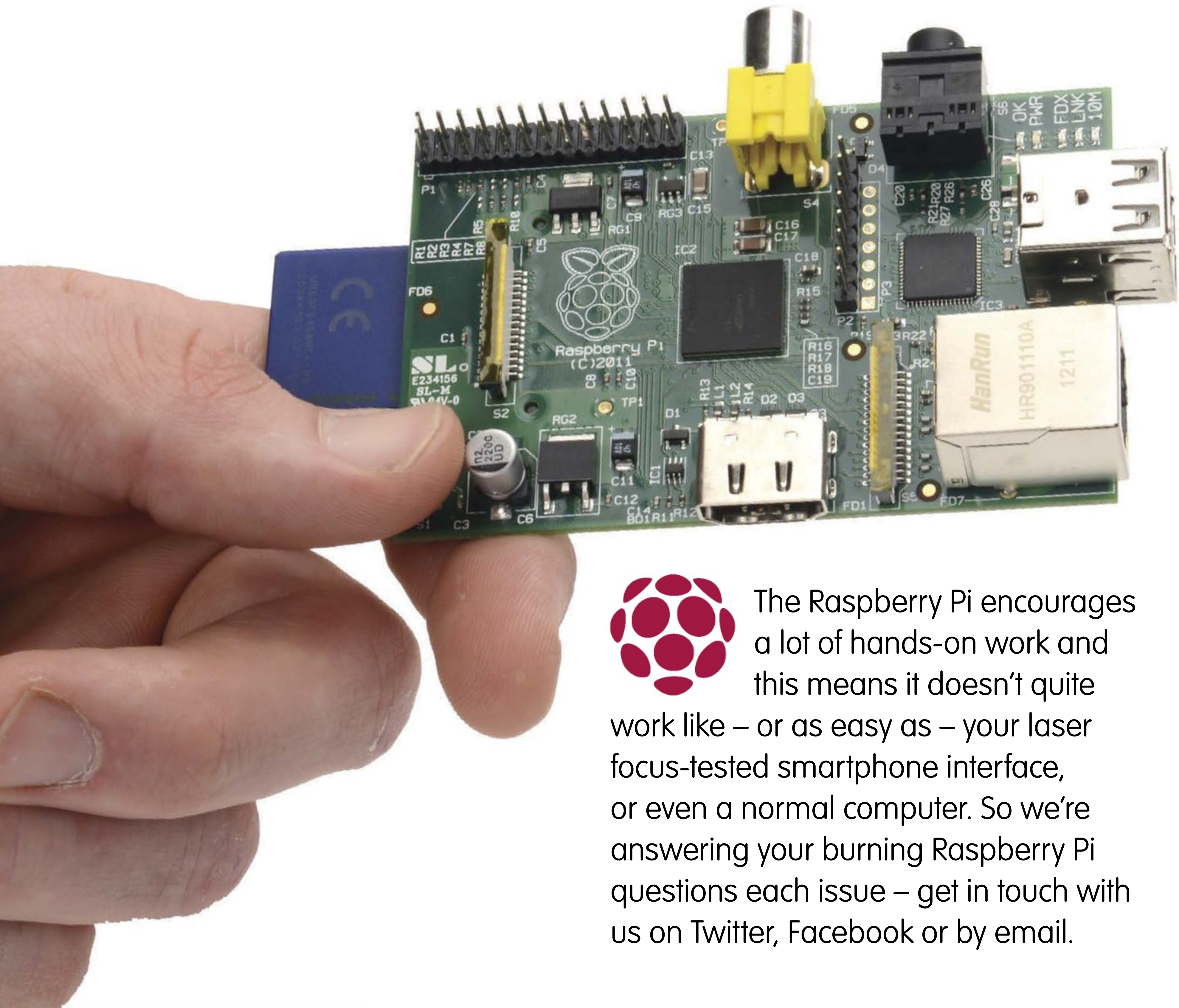
Short memory

The RasPi's size and price point dictates that it doesn't have a lot of memory. How frugal are you with your code?

Talking Pi

Join the conversation at…

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easy as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

I heard there's a new browser for the Raspberry Pi, how do I go about getting it?
Alistair via Facebook

Yes, the new Epiphany-based browser for the Raspberry Pi is an excellent piece of software that is much faster than the current offering included with the Raspberry Pi. The latest version of Raspbian contains the browser, so if you're setting up or doing a fresh install you will already have the brand new browser. Otherwise, you can install Epiphany in the browser by updating the software and performing a distribution upgrade by using the following three commands in the terminal:

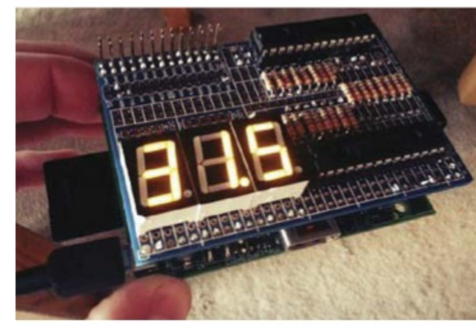
```
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo apt-get install epiphany-browser
```

I don't really like the Epiphany browser on other operating systems, can I still use Midori?
Jill Burton via Facebook

If you're using a slightly older version of Raspbian then you don't need to worry – simply updating the software won't remove Midori or even install Epiphany at all. If you're creating a new Raspbian SD card from the downloads on the Raspberry Pi site then it won't have Midori but you can re-install it if you wish. Do this with the following:

```
$ sudo apt-get update
$ sudo apt-get install midori
```

Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag



AverageManvsPi: #RasPiMag pic.twitter.com/dUMaecOpHA

windfallprophet: Twitter activated shot dispensing bot! #RaspberryPi + #Bartendro project made for a friends bar. #ShotBot #RasPiMag pic.twitter.com/Kk4cdEPUqR

I like the idea of the Kano computer system but I already have a Raspberry Pi. Can I make one?
John Hogan via email

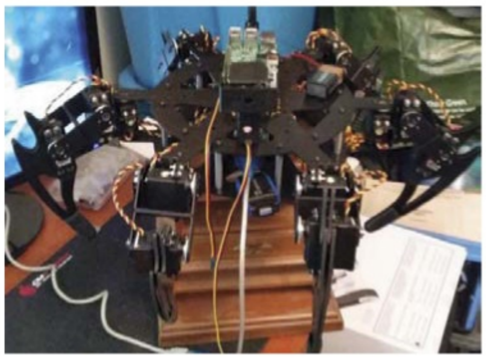
Yes you certainly can. There are a couple of things that make up the Kano system and that's the hardware and software. The hardware is cool but the operating system is what really makes it useful; luckily that's free for everyone to use and you can grab it straight from the Kano website, **kano.me**.

Write it to an SD card like any other image, get PDFs for how to use it and you're on your way.

Am I stuck to only using an ethernet cable on my Raspberry Pi to go on the internet?
Dan Chan via Facebook

You can use Wi-Fi on the Raspberry Pi like just about any other PC or Laptop, although you'll need to buy a USB Wi-Fi dongle to do this. It's easier on the Model B+ as there are more USB ports, but on the original Raspberry Pi's you may need to get a powered USB hub if you want to have mouse, keyboard and wifi connected at once.

Not every wireless dongle works though. A full list of tried and tested dongles can be found online here: **http://elinux.org/RPi_USB_Wi-Fi_Adapters**



theuberchad: Almost ready to have the scorpion bot controlled by its #RaspberryPi brain. #raspimag pic.twitter.com/cqtwafgpxM

artekw: Wireless Home Automation Base station with RaspberryPi & Arduino #RasPiMag pic.twitter.com/hSm90i5KZr

CodyErekson: Meet @RoboPaulLives, my #raspberrypi powered robotic psychic octopus. :) pic.twitter.com/KxDaxZ90Il

Next issue

Get inspired · Expert advice · Easy-to-follow guides

Supercharge your Raspberry Pi

Plus Wireless access point, networking peripherals and more

Get this issue's source code at:

www.linuxuser.co.uk/raspicode